Optimizing Synchronization Cost for Mobile Devices: The Expedient Trickle Sync Algorithm

by

Brad J. BARCLAY

B.Sc. (Hon.), Brock University, 1999

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Computer Science Department

University of Victoria

Optimizing Synchronization Cost for Mobile Devices: The Expedient Trickle Sync Algorithm

by

Brad J. BARCLAY
BSc, Brock University, 1999

Supervisory Committee

Dr. Jens. WEBER, Supervisor (Computer Science Department)

Dr. Bill. WADGE, Member (Computer Science Department)

Dr. Yvonne. COADY, Member (Computer Science Department)

Supervisory Committee

Dr. Jens. WEBER, Supervisor (Computer Science Department)

Dr. Bill. WADGE, Member (Computer Science Department)

Dr. Yvonne. COADY, Member (Computer Science Department)

Abstract

In this thesis, I propose an algorithm for optimizing the cost involved with synchronizing the data contained in mobile wireless devices, named *Expedient Trickle Sync* (ETS). In this thesis, I focus on two significant cost factors: firstly, that of the cost of transmitting information across the network, and secondly, the cost associated with user access to out-of-date information. The ETS algorithm attempts to balance these two cost factors via a simple set of heuristics which calculate at regular intervals a test value, based on a variety of observations, and a threshold value which is recalculated on a daily basis. Additionally, the ETS algorithm will prioritize records based on their probability of access and the cost associated with synchronizing them, thus possibly deferring the synchronization of records with a low probability of access until it is possible to resolve their replication in a lower-cost network environment. This thesis evaluates the ETS algorithm alongside other optimistic replication algorithms in a custom developed simulation environment, the results of which show that in many use scenarios, the ETS algorithm can indeed reduce the cost of data synchronization.

Table of Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **iv**

List of Tables **viii**

List of Figures **x**

Acknowledgements **xii**

1 Introduction **1**

1.1 Introduction 1

1.2 Structure of Thesis 3

2 Background **5**

2.1 Introduction 5

2.2 Synchronization Topologies 7

2.3 Connection and Authentication 9

2.4 Agreement on Databases to Synchronize 12

2.5 Selection of a Database to Synchronize 13

2.6 Mapping of Records on Both Devices 15

2.7 Identification of Modified Records 17

2.8 Identification of Conflicting Records 23
2.9 Conflict Resolution 25
2.10 Determining what information to include in the update packet 27
2.11 Writing Updated Records and Removing Deleted Records 30
2.12 Updating Synchronization Metadata and Disconnection 31
2.13 Conclusion 33

3 Related Work **35**
3.1 Introduction 35
3.2 Set Reconciliation 36
3.3 File and Database Replication 38
3.4 Optimistic Replication 40
3.5 Clock Synchronization 40

4 Simulation Environment **42**
4.1 Introduction 42
4.2 Simulated Elements 43
4.3 Models 47
4.4 Controllers 52
4.5 Comparative Protocols 56
4.6 Limitations 58

5 Expedient Trickle Sync Algorithm **62**
5.1 Introduction 62
5.2 Determining when to synchronize 63
5.3 Determining which network to synchronize with 79
5.4 Determining which records to synchronize 80
5.5 Prioritizing synchronization order 81
5.6 Comments 82

6 User Model Study **83**

6.1 Purpose 83

6.2 Methodology 84

6.3 Results 85

6.4 Conclusion 86

7 Experiment and Results **87**

7.1 Introduction 87

7.2 Experimental Data 87

7.3 Results 95

7.4 Analysis of Results 117

8 Conclusions **122**

8.1 Future Research 122

8.2 Concluding Remarks 126

Bibliography **128**

A Simulation Library Sources **135**

B Glossary of Algorithms, Equations, Symbols, and Variables **136**

C Survey Invitation to Participate **142**

D Letter of Implied Consent **144**

E User Model Study Questionnaire **148**

F Simulation XML Property Files **150**

F.1 Simple User A 150

F.2 Simple User B 150

F.3 Business User A, C 151

F.4 Business User B 152
F.5 Mobile Salesperson A 153
F.6 Truck Driver A 154

G Raw Result Data **155**

G.1 User Model Protocol Comparisons 155
G.2 ETS algorithm evaluations for varying values of k 157

List of Tables

7.1 Common network properties used in the evaluated User Models . . . 88
7.2 Basic Properties of Simple User A 89
7.3 Networks available to Simple User A by location 89
7.4 Database properties for Simple User A 89
7.5 Networks available to Simple User B by location 90
7.6 Basic Properties of Business User A 90
7.7 Networks available to Business User A by location 90
7.8 Database properties for Business User A 91
7.9 Basic Properties of Business User B 91
7.10 Basic Properties of Mobile Salesperson A 93
7.11 Networks available to Mobile Salesperson A by location 93
7.12 Database properties for Mobile Salesperson A 94
7.13 Basic Properties of Truck Driver A 94
7.14 Networks available to Truck Driver A by location 94
7.15 Database properties for Truck Driver A 95

G.1 Raw Results for Simple User A 155
G.2 Raw Results for Simple User B 155
G.3 Raw Results for Business User A 156
G.4 Raw Results for Business User B 156
G.5 Raw Results for Business User C 156

G.6 Raw Results for Mobile Salesperson A 156
G.7 Raw Results for Truck Driver A 157
G.8 Raw Results for Simple User B when evaluating the ETS algorithm at different values of k 157
G.9 Raw Results for Mobile Salesperson A when evaluating the ETS algorithm at different values of k 157

List of Figures

2.1 Synchronization Network Topologies 10

5.1 How modifying the threshold could negatively impact cost 67
5.2 An example of a series of threshold evaluations 73
5.3 Two possible ways to have equal cost points without making progress 74
5.4 An example of all three points in the threshold set having equal cost . 75
5.5 An example where the midpoint lands on a local maxima 76
5.6 An example where a random use event causes a cost spike 77
5.7 An example demonstrating narrow and broad minima 79

7.1 Graph displaying the results for Simple User A 98
7.2 Graph displaying the daily ETS costs for Simple User A for a 100-day run 99
7.3 Graph displaying the results for Simple User B 101
7.4 Graph displaying ETS results for Simple User B, at different values of k 102
7.5 Close-up detail displaying ETS results for Simple User B, at different values of k 103
7.6 Graph displaying the daily ETS costs for Simple User B for a 100-day run 104
7.7 Graph displaying the results for Business User A 106
7.8 Graph displaying the daily ETS costs for Business User A for a 100-day run 107

7.9 Graph displaying the results for Business User B 109
7.10 Graph displaying the daily ETS costs for Business User B for a 100-day run 110
7.11 Graph displaying the results for Business User C 112
7.12 Graph displaying the results for Mobile Salesperson A 114
7.13 Graph displaying ETS results for Mobile Salesperson A, at different values of k 115
7.14 Graph displaying the daily ETS costs for Mobile Salesperson A for a 100-day run 116
7.15 Graph displaying the results for Truck Driver A 118
7.16 Graph displaying the daily ETS costs for Truck Driver A for a 100-day run 119
7.17 Graphs comparing Fast Sync to ETS at 1 Standard Deviation 120

Acknowledgements

This thesis has been the culmination of over ten years of study into the area of data synchronization for mobile devices, started during my undergraduate years at Brock University in St. Catharines, Ontario, through my time working for IBM, and with the OpenTAPAS Project, until today – sometimes with support, other times purely of my own volition. As such, there are a number of people who rightfully must be acknowledged here, for without their aid, this body of work may have never come to pass.

Firstly, I'd like to acknowledge Prof. Jon Radue of Brock University in St. Catharines, Ontario, who acted as my undergraduate supervisor in what became *The jSyncManager*. Without the jSyncManager, I wouldn't have been writing this thesis today, as its genesis is firmly rooted in the success of this project.

Secondly, I'd like to thank Dr. Morgan Price, who initiated what became the OpenTAPAS Project. It was through Morgan's need for a data synchronization solution that led him to me, and to eventually bring me on board the OpenTAPAS Project as their mobile device data synchronization expert. We had great ideas and a great team, all of which was brought down by politics outside our control.

It was through Dr. Price that I was introduced to Dr. Jens Weber, who would become my graduate supervisor, and an indispensable hand in completing this thesis. Dr. Weber has been the lens through which the ideas presented in this thesis were able to come into better focus. It was Dr. Weber who stood up for me when I decided to go back to school to complete my Masters degree, and for this I will forever be grateful.

No acknowledgement is complete without a nod to the family support structure behind the researcher. As such, I'd like to thank my mother and father, Pauline and Robert Richard (Dick) Barclay for their love and support throughout my life. This work is every bit as much a testament to their efforts as it is to my own. Special acknowledgements go out to my brother and only sibling, David Barclay, not only for the life-long bond between us that is the equal of no other, but for helping me design the outline for the Mobile Salesperson A user model. Dave, you may never have the interest or opportunity to explore education in the manner that I have, but you've made your mark on science, and it will forever be recorded in these pages.

Finally, extra special acknowledgements go out to my wife, Gülcan. My love, a phase in our lives is about to pass, and I look forward to nothing more than exploring all of our remaining ones together. You have been my rock throughout the writing of this thesis, which I herein dedicate whole-heartedly to you.

Chapter 1

Introduction

1.1 Introduction

Mobile, semi-connected computing devices have quickly become one of the most dominant forms of personal data storage, data retrieval, and communications tools on the planet. Convergence has blurred the lines between devices which have traditionally focussed on only a few areas of use, such as computation, data storage/retrieval, entertainment, and communications, to the point where many mobile instruments have features of all the aforementioned in them. Mobile phones in particular can provide telephony, e-mail, instant messaging, music and video playback, games, imaging, and personal/business databases. Such devices can run sophisticated software packages, and can often be used in the place of bulkier personal computers such as laptops, and are often desirable in situations where portability, mobility, locality, and interaction with other devices are important [52, 58].

Unfortunately, at this point in time network availability is not ubiquitous. Cellular devices often have issues connecting to a suitable network while inside buildings, under overpasses, inside tunnels, and in sparsely populated locales. Wireless devices in general do not provide as much bandwidth or reliability as a wired network connection [47, 21]. For these reasons, applications written for mobile devices, particularly those which rely on some form of database, typically require local storage to ensure

data availability when network service is inaccessible [30]. Keeping this information up-to-date with respect to other databases requires the replication of data between devices, a task known as data synchronization.

Data synchronization for most mobile devices is still fairly primitive. It may require the user to plug their mobile device in to a more traditional personal computer style device via a cable or docking station, and even when a wireless connection for synchronization is available, the user usually must initiate the synchronization process manually. The device is typically unusable until the synchronization process has completed. This may be acceptable for small databases which dont change frequently (such as a personal address book), however for more complicated applications in business, government, and medicine, the requirements to be physically present with the device to be synchronized against (or potentially through), to manually enable and initialize the synchronization process, and to wait for the process to complete may be onerous, and may work against the desire to ensure that the synchronized data is as up-to-date as is reasonably possible at the time the data is required. For large data sets which change frequently, these requirements can become problematic, and can result in the user making decisions based on invalid data, which in the long term can cause the user to lose trust in the information the device contains, reducing the perceived utility of the device. This can lead some users to stop using their mobile device for data storage. Lengthy and onerous synchronization procedures may result in the user becoming lax in ensuring they synchronize frequently, increasing the chances that a cost may be incurred due to decisions based on invalid, out-of-date data [40].

Fortunately, mobile devices are becoming sufficiently powerful that they can employ intelligence coupled with automated background synchronization to avoid these issues. However, such intelligent synchronization systems have thus far not been implemented, in favour of the user-driven status quo.

Another recent innovation in many mobile devices is the ability to connect to multiple networks. Mobile phones such as the Apple iPhone and Palm Treo are capable of connecting to cellular (GPRS, EDGE) and WiFi (802.11.a/b/g) networks for data transfer. Each of these different network types typically have completely different data transfer rates and costs associated with them, with even identical networking technologies having different costs at different locales. These differences can also affect the cost associated with data synchronization.

This thesis proposes an improved mechanism for synchronizing data with such devices, named Expedient Trickle Sync (ETS). ETS is an intelligent, automated data synchronization system that synchronizes in the background, with the goal of reducing the overall cost of data synchronization in the following two areas:

1. By reducing the real cost associated with network access, and

2. By reducing the virtual cost associated with the use of invalid or out-of-date data.

Verification of the effectiveness of this algorithm was performed by developing a simulation environment for evaluating the average cost of a series of synchronization algorithms, by running each one through the simulation environment against a series of user model profiles. The ETS algorithm was tested against both existing industry-standard algorithms, and a pair of control algorithms.

1.2 Structure of Thesis

The ensuing chapter provides a survey of existing research in topics surrounding data synchronization and replication, and derives a list of base requirements for semi-connected devices to optimistically replicate their information with one or more other devices. Chapter 3 discusses the simulation environment that was developed for evaluating the synchronization protocols tested, along with the various mathematical

models used in developing the simulation environment, and limitations of the same. Chapter 4 details the design of the ETS algorithm itself, with focus on intelligently determining when to synchronize, what networks to synchronize via, which records to synchronize, and prioritizing the record replication order. Chapter 5 details my attempts at a user model study, so as to better create user models to feed into the simulation environment. Chapter 6 details the experiments run through the simulation environment, along with their results, and an analysis of the obtained results.. Chapter 7 concludes this thesis with a discussion of possible future research directions, and my concluding remarks.

Chapter 2

Background

2.1 Introduction

People have long desired the ability to carry important information with them, as an aid to memory, as a portable resource, for entertainment, or to transport data from one location to another. Throughout history, humanity has devised a variety of mechanisms for storing and transmitting data.

With the rise of the computer era, this desire for portable data access has continued; initially in the form of portable storage media such as the compact cassette and diskette, but more recently in the form of portable electronic devices which can act as both storage and access mechanisms, such as laptops, PDAs, cellular telephones, and portable media devices such as Apples iPod. Initially, due to cost, size, and lack of infrastructure, these devices contained no wireless access capabilities, and thus required a wired connection to a network or desktop computer to be filled with existing digital information. As this information changed over time on one device or another, the need to synchronize the two became imperative.

Even in the modern era, where wireless facilities are common in first-world countries, it is often still desirable to be able to physically carry data with us, as opposed to being able to access it over a network. Cellular service may be sporadic at best inside buildings and vehicles. WiFi connections only cover short distances, and thus

lack coverage in all but a few locations. Wireless network signals can be blocked, and downtime can be encountered [47]. Firewalls may block data access from certain locations, and certain locales may prevent access to certain types of data services by way of legislation. As well, in under serviced areas, network contention and delay may be high, resulting in long wait times for desired data. With the continued presence of these issues, in many problem domains it remains desirable to be able to maintain a local repository of data on the device, with synchronization services when a network connection is available to effect modifications to both local and remote data repositories.

Due to a lack of guaranteed connectivity, as well as a desire to permit databases to be updated on two or more synchronized devices, these devices typically use optimistic replication [14, 51, 49] for general synchronization services. As we generally make no assumptions as to the types of data stored on such devices, pessimistic replication [14, 51] can be difficult to implement; if the set of records which might be updated on the mobile device is unknown at synchronization time, record-level locking becomes impossible when the device lacks a network connection [33]. Optimistic replication has been shown to permit improved data accessibility in ad-hoc networks, such as those frequently used by mobile devices [26].

For the purposes of this chapter we denote devices of this type to be semi-connected devices, wherein while they do provide some form of network connectivity, its presence at any given time t is not guaranteed, and that P_t, the probability that at time t the network is available, is lower than would be expected for a stationary device that is wired into a dedicated network connection.

The goal of this chapter is to describe the set of base requirements for optimistic replication when using semi-connected devices, based on my experiences surrounding my work developing the jSyncManager [11, 12], a synchronization toolkit for PalmOS based devices, along with existing published research in relevant areas. It will

discuss synchronization topologies, mobile device connection and authentication issues, agreement on databases to be synchronized, selection of a specific database to synchronize, the mapping of records on both devices, the identification of modified records, identification of potentially conflicting records, conflict resolution, determining what information to include in an update packet, the writing of updated records and the removal of deleted records, updating synchronization metadata, and mobile device disconnection.

2.2 Synchronization Topologies

Two very important considerations for designing a synchronization system are determining how mobile devices and servers are to exchange data, and which systems in the synchronization network each device can connect to for a synchronization session. For the purposes of this paper, the set of all devices that can synchronize a specific database will comprise the synchronization cloud.

The topology of a synchronization cloud is akin to the standard network topologies. Common such topologies, and how they are reflected in a synchronization cloud, include [55]:

1. **Star**: (fig. 2.1) in a typical star synchronization network, each mobile device connects solely to a centralized server. Data modified on one mobile device propagates to any other mobile device through two synchronizations (mobile to server, server to mobile). In this way, the synchronization strategy is one-to-one for the mobile devices, and one-to-many for the centralized server device.

2. **Ring**: (fig. 2.1) in a ring synchronization network, each mobile device and/or server in the cloud synchronizes with two other devices in the cloud. The devices in the cloud chain together in this way, with the last device in the chain synchronizing against the penultimate device in the chain and the first device in the chain. While the synchronization rules can be relatively simplified (as

each device only synchronizes against two known, fixed devices), it may take up to $n/2$ synchronization sessions for a piece of data to propagate through the network. In addition, adding new devices to the network can be highly disruptive, requiring a portion of the chain to be broken, and the new device inserted. The data synchronization in this sort of network is easily broken if one mobile device in the system is not synchronized for a lengthy period of time, and devices adjacent to each other in the ring require that their users are frequently in close proximity to each other to permit the synchronization. For these reasons, ring topologies are generally unknown in practical use. Ring topologies require a one-to-many synchronization strategy for all devices in the cloud.

3. **Tree**: (fig. 2.1) in a tree synchronization network, each mobile device synchronizes with single, fixed device (typically a server). These servers, in turn, may synchronize against higher-level server, eventually leading to a single root server. In the tree model, the mobile devices are the leaves of the tree, with the servers making up the root and inner nodes. Leaves synchronize using a one-to-one strategy, inner nodes and the root node synchronize downwards (towards the leaves) in a one-to-many strategy. Inner nodes synchronize upwards in a one-to-one strategy. This model is best when the synchronization cloud is geographically distributed; various subsets of the mobile devices synchronize against their own local servers, and these local servers synchronize against high-level servers, and eventually with the root server, which disseminates modifications downwards throughout the cloud. In a worst-case scenario, the dissemination of data between any two mobile devices in the cloud will be the length of the root paths between the two most distant leaves in the tree. However, in usage scenarios where the locality of data is important, the number

of synchronizations required for local data updates between two mobile devices will typically be two.

4. **Complete**: (fig. 2.1) in a complete synchronization network, any member of the synchronization cloud is permitted to synchronize directly against any other member of the cloud. Thus each device employs a one-to-many model of synchronization. In a best-case scenario, a modification made to a given mobile device can be effected to any other mobile device in a single synchronization. However, in practice it may take a significant amount of time for a change to propagate its way through the entire cloud, and if a subset of devices only ever synchronize amongst themselves, changes made within that subset may never propagate to other devices in the synchronization cloud. One solution to to this problem is to ensure that all servers synchronize against all other servers on a regular basis, and that all mobile devices in the cloud synchronize with at least one server on a regular basis.

5. **Hybrid**: (fig. 2.1) several hybrid approaches may be taken. One common hybrid approach has all of the mobile devices acting as nodes in a star network, which is centred with either a ring network or complete network of servers. In this way, the mobile devices synchronize against one (or more) servers, and the servers synchronize against each-other. In such hybrid models, the mobile devices may be permitted to synchronize with any server, or may be fixed against which server they may synchronize with.

2.3 Connection and Authentication

The first requirement to be identified is the ability to connect the mobile device to the synchronization host. How the connection is made can vary greatly between different classes of devices, with varying protocols and connection methods employed. Mobile

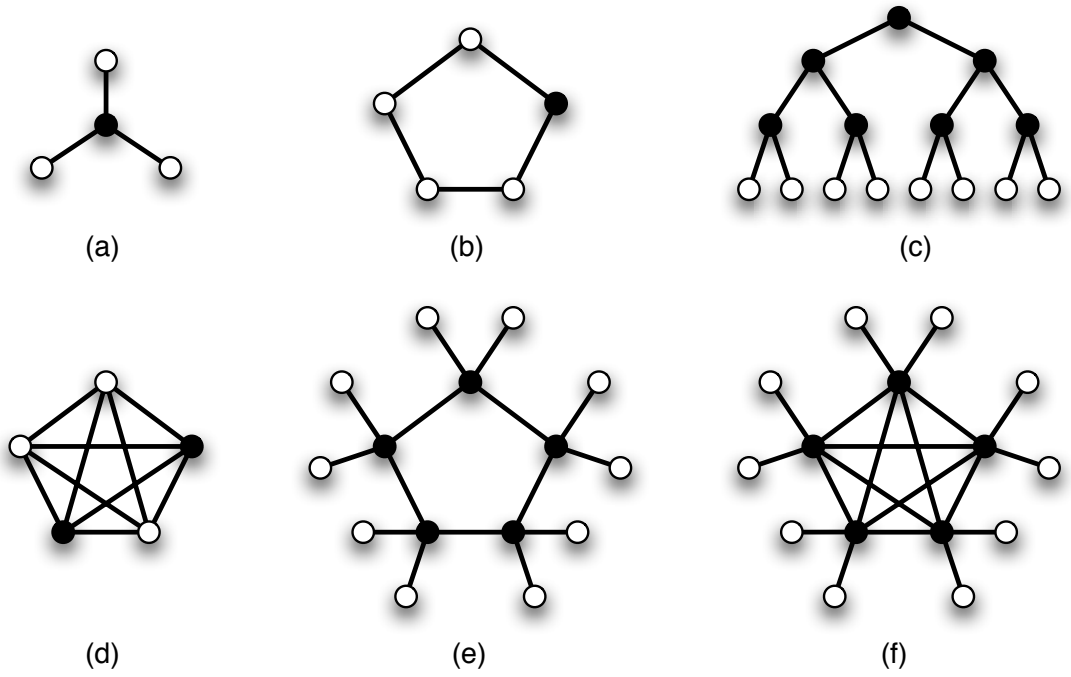

Figure 2.1: Synchronization Network Topologies. White nodes denote mobile devices, black nodes denote servers. (a) Star. (b) Ring. (c) Tree. (d) Complete. (e) Hybrid Star/Ring. (f) Hybrid Star/Complete

phones may use Bluetooth or USB connectivity to a host, or may use a TCP socket encapsulated within a General Packet Radio Service (GPRS) or Enhanced Data rates for GSM Evolution (EDGE) wireless connection. WiFi through 801.11b/g is also an option on some portable phones. Handheld devices, such as those from Palm Inc., may connect using Bluetooth, USB, RS-232 serial, infrared, modem, or WiFi connectivity using a custom protocol stack (HotSync), either directly or encapsulated within TCP packets.

Once a network connection of any type has been made, it is typically important to authenticate the device against the host in some manner, to ensure that the device and its owner are permitted access to the system in question. Oftentimes in situations where devices are connected in a physical manner proximal to the synchronization host, this step may either be skipped, or simply used to verify whether

or not the given device has synchronized with this system previously. Devices that synchronize through proximity wireless systems such as Bluetooth may simply rely on a pre-existing Bluetooth pairing of the host and the semi-connected device for the purposes of authentication. WiFi enabled devices may also contain no explicit authentication, requiring the user to configure as appropriate either WEP or WPA style WiFi authentication.

Such simple authentication mechanisms are common on most semi-connected device types, outside certain specialized device types [22]. Common commercial handhelds and mobile telephones lack any form of authentication outside authenticating against the network itself. Perhaps more important is host or database level access authentication. An important consideration in designing an authentication mechanism is the fact that semi-connected devices are portable by their nature, typically relatively valuable, and are both targets for theft and loss. Under such circumstances, it is desirable to prevent an unauthorized user of the device from being able to either access the existing data on the device, or to be able to connect to the synchronization host to retrieve more data (or to potentially corrupt data within the hosts database by purposefully synchronizing in bad information). While access to the existing data on the device needs to occur outside the scope of the data synchronization process (and is thus outside the scope of this paper), access control to the synchronization host thus requires the ability to both authenticate the device and to authenticate the user of the device. In situations that require high levels of data security, authenticating both becomes important as it both prevents unknown users from accessing or modifying data on the synchronization host, and as it prevents authorized users from substituting their own devices.

Device and user authentication may take several forms. Device authentication may be performed via hardware identifiers (*e.g.* CPU or system serial codes), via cryptographic certificates, or a combination of the two. User authentication is typ-

ically best performed using an external factor authentication, such as a password, biometric validation, or through proximity to a personal security device carried by the user.

Once authenticated, various levels of data access may be granted or barred based on the credentials of the device and the user. Access may be granted only to specific databases, or it may affect which records inside a given database are acceptable to transfer. As well, once authenticated, handshaking may occur to enable data encryption for all further synchronization requests and responses during the session, so as to keep data inaccessible to third parties.

Other forms of handshaking and global data synchronization may also occur as part of the connection phase, such as device capabilities an characteristics, which may affect how synchronization is to proceed. Very basic data synchronization tasks such as time synchronization, which do not necessarily belong to any specific database or type but which are of global use, may also occur as part of the connection phase.

2.4 Agreement on Databases to Synchronize

Once the connection has been initiated and authenticated, an agreement needs to be arrived upon to determine what databases between the semi-connected device and the synchronization host need to be synchronized. Part of this agreement may involve connecting the mobile device to a remote database, routed through the synchronization host, and any associated authentication required between the synchronization host and the remote database, the mobile device and the remote database, or both. Selection of the databases to synchronize is dependent upon the application support on the handheld, the databases available through the synchronization host, any authorization rules and permissions in place, and the ability of the mobile device and/or the synchronization host to synchronize the data types exposed by those databases. Only when all of these criteria are met can a database be synchronized between the

device and the host. Thus, the selection of the databases to be synchronized is the subset of all databases available on the device and on the host that fit the above requirements.

There are a variety of ways in which the device and the host can determine which elements fit into this subset. This may be achieved through a handshaking phase where the two devices agree upon the subset of databases to synchronize, or it may be driven primarily by one of the systems involved. The later situation is common for PDA-style devices, where plug-in units called conduits [22, 12] are run in sequence at synchronization time, each testing the mobile device to see if the data they wish to synchronize is available. If it is not available, the conduit can either exit gracefully (permitting the next conduit in sequence to run), or construct and initialize a new database to use for synchronization purposes [12].

Note that while this scenario has the advantage of the mobile device not needing to have the database to be synchronized at connection time, it has the disadvantage of requiring that the necessary conduit be installed at connection time on the host computer. As such, while the presence of a conduit on the host system can permit synchronization where the portable database is missing, the presence of the database on the mobile device alone is insufficient to permit synchronization.

2.5 Selection of a Database to Synchronize

Once the subset of databases to be synchronized has been agreed upon, one of these databases may be selected for synchronization. There are a variety of mechanisms for selecting which database should be synchronized next at any given time. The simplest mechanism is to simply iterate through the list of databases in whatever order they occur in the subset of databases to be synchronized. Using this mechanism doesnt require any computation and thus selection is fast: at the beginning of synchronization, select the first database in the set, and iterate through the list

thereafter. This approach may encounter problems in situations where a single application has multiple database to be synchronized, as there may be a necessary order for the databases to be synchronized. For example, it may be necessary to ensure that a common encryption key database be properly synchronized before the databases which rely on this key are synchronized.

A refinement to this approach is to use an application-centric view of synchronization, where an entire application consisting of multiple databases is synchronized as a single unit. In this situation, while the order of the applications is not guaranteed, the order of the databases synchronized as part of that application can be guaranteed. This model is easily exposed when using host-side conduits for synchronization control, as each conduit can manage the databases it needs in whatever order it desires, however there is no guarantee as to what order each conduit is giving the opportunity to exchange data.

This latter model can impose efficiency penalties under certain circumstances. In the case of a database which is shared amongst multiple applications, each conduit accessing the database may either force it to be synchronized multiple times needlessly, or at the very least needs to check to see if the data has been synchronized by a previous conduit and skipped.

An enhancement that gets rid of the need to check whether a database has already been synchronized during an existing session is to implement the notion of priority to the database selection. Using this system, the database synchronization order can be determined by ordering the databases by a stored priority value, synchronizing the database with the highest priority first, and the database with the lowest priority last. Under such a system, any shared databases that require synchronization prior to applications which rely on them can be given a higher priority, and thus are guaranteed to be synchronized earlier in the process.

The use of a priority field comes with its own set of difficulties. Firstly, the handling of databases that have the same priority cannot be done in any guaranteed order; it is usually a documented limitation of such systems that there is no guarantee of ordering between databases with the same priority. Secondly, it is possible for application developers to abuse the priority ordering by simply marking themselves as the highest priority level. This can be mitigated by allowing an administrator control of the priority levels for each database, using the application-defined priority as a default value.

2.6 Mapping of Records on Both Devices

Once a database has been selected for synchronization, it is up to the synchronization system to determine (or provide) a mechanism for mapping the records from both the host and the mobile device. As each system may store this information in differing formats, it can be very difficult to know that record x on the mobile device mates up with record y on the synchronization host. This is complicated by the assumption that the data may differ between the two devices (thus necessitating the synchronization process). It is further complicated by the fact that the host and the mobile device may use different identification values for the same record. However, even with these challenges present, it is necessary to have the ability to match a record between the mobile device and the synchronization host in order to determine whether it requires synchronization, which fields need to be synchronized, and to ensure that the correct record(s) are updated on both devices.

The mechanism used for determining the record mapping can affect the time taken to perform the synchronization task. Linear scanning of the records from one system containing n records against the other system containing m records, using a suitable matching algorithm between the two systems would require on average nm/2

comparisons. This could be lengthy for very large databases, and may easily exceed the users patience in such situations.

The simplest solution to the mapping problem is to ensure that both the mobile device and the synchronization host use the same identification numbering system for records. An unsigned 32-bit value is usually sufficient for mobile device synchronization, as it permits over 4.2 billion unique record IDs. Dividing this address space in half can ensure that new records created on both the mobile device and the synchronization host between synchronization sessions do not overlap.

However, it is often the situation that either one or both sides of the synchronization solution are developed by different groups or even organizations: while one may be in control of the host-side database, they may be synchronizing against a vendor-supplied mobile database that uses a differing identification system. Or one may find oneself in the situation of wanting to bring mobile device synchronization into an organization which has entrenched host-side applications from third-party vendors, for which you do not have control of the database. Additionally, the mobile devices operating system may impose its own record identification scheme (cf: PalmOS).

In situations where control over the unique identifiers on one or both devices exists, maintaining identical identifiers on both ends of the process may be difficult or impossible. In such a situation, another mapping mechanism needs to be developed. One such mechanism is to maintain a mapping database for conversion between one identification scheme and the other. Depending on the synchronization needs, this can be implemented as a hash, with one sides identification scheme serving as the key, and the remote ends mapping serving as the value associated with each key. Maintaining a second hash for the reverse direction may also be desirable in a variety of situations. Keeping two such hashes in proper synchronization with each other becomes an important added task, as does persistent storage of such a map, for use during subsequent synchronization sessions.

Finally, in situations where one has no control over either the host systems or the mobile systems record identification mechanism it may be impossible to maintain a unique record identifier (often because one or both sides uses a third-party application which doesnt maintain a unique identifier for each data record). In such situations, one of the records fields may need to serve as a unique identifier, with a matching heuristic used to find the matching record (if any) on the other devices database. Such a situation is typically undesirable, as scanning all the records to build the mapping at synchronization time may require significant runtime. This process can also be problematic in situations where multiple records share the same unique identifier field (such as two calendar appointments for the same date and time), or where the unique identifier field for a record on one of the devices is modified (such as the name of a contact in an address book). In such situations, the mapping heuristic may need to take other record fields into account to attempt to find the correct mapping. Note however that if a record on one end of the synchronization is sufficiently modified, finding this mapping at synchronization time may become impossible for the system to do in an automated way, necessitating resolution by the user.

2.7 Identification of Modified Records

Assuming a mapping of records between the mobile device and the host database selected for synchronization is possible, it is important for us to be able to identify specifically those records on both devices which have been modified since the last synchronization session. For the purposes of this paper, we define a modified record to be one which falls into one of the following categories:

- **Added**: Records that have been newly added to one of the databases since the last synchronization session between the two devices;

- **Updated**: Records that existed at the time of the last synchronization, but which have one or more fields which have been modified in either database since the last synchronization session;

- **Deleted**: Records that have been removed from one of the databases since the last synchronization session.

Identification of the modified records is an important facility [21], as in an attempt to minimize the synchronization time and minimize the amount of data that needs to be transferred, we need to separate the records that have been changed since the last synchronization session with the given host from those which have remained identical [29, 44]. Records which have not changed between synchronization sessions on one or both devices do not need to be synchronized, and thus require no records to be updated on either device.

Modified record identification can be most simply implemented using a single per-record flag that is set whenever a record has been modified. If these flags are cleared after every synchronization session, any records so flagged will have been modified since the last synchronization.

An alternate system is to always record a modification date/time for each record, as well as the date/time for each synchronization. Using this system, we can select only those records with a modification date since the last synchronization date. Note that while this system is relatively easy to implement and process, it becomes very important that the clocks on all devices within the synchronization cloud are themselves properly synchronized to ensure that times generated by different devices can be properly compared.

Another well documented system for identifying modifications is through the use of a *version vector*. Version vectors are a compact representation of the update history of a replica of a record, and can thus be used to both determine which records are modified, and which are in conflict [21].

Identifying records using any of these methods does introduce some issues, however. In particular, once a modified record has been identified using this scheme, it is necessary to identify which of the three types of modifications has taken place (added, updated, deleted). Added records are easy to identify if we have a good mapping system in place to match records between the device and host; added records will be those which exist on one device, but which have no mapping to any records on the other device. It is worth noting that in the case where mapping occurs via field comparison heuristics, added records may be mistaken for existing records if the identification fields correspond sufficiently closely. That is, if record A2 (record 2 on device A), a newly added record, is heuristically matched to record B1 (which normally maps to record A1), the later update phase will run its update routine on B1, modifying it to an incorrect state. Later in the synchronization (or in a subsequent synchronization session), the system may determine that B1 and A1 now differ, propagating the bad record data back to the original device as a different record. For this reason, if heuristic record mapping is to be used, it is important to ensure that it has a low probability of mismatching records.

Identification of deleted records can be complicated if the record on the system where it has been deleted is erased (that is, where no record of it having existed on the system and its status as having been deleted has been recorded). In this case, a deleted record on device A may in fact be mistaken as a new record on device B (as device B contains a copy of the old record which is now no longer present on A), thus causing deleted records to re-appear after every synchronization session. As such, a mechanism for detecting deleted records is necessary.

Identifying deleted records can take several forms. In one algorithm, we do not erase between synchronization sessions, but instead flag them as having been deleted. In this case, it is important for applications using the database to simply ignore deleted records. At synchronization time, deleted records on each device can be

identified, with the deletions synchronized between hosts. To conserve space on mobile devices when using this algorithm, it may be desirable to remove the data for the record, maintaining just the records metadata until synchronization time. Deleted records may be truly erased once the synchronization session for the database is complete, as part of the post-synchronization cleanup (for two-way synchronization), or may continue to be flagged for deletion and erased once all hosts have flagged the record as deleted (for n-way synchronization).

In another algorithm, we make use of a record identification map to find records that are in the map, but which do not exist in the target database; if they are in the map, they existed in the database during the last synchronization session, and if they no longer exist, they must have been deleted in the intervening time between the last synchronization and the current synchronization. The record can then be deleted from the other device. This deletion can generally be total erasure. Once the deletion is complete, the mapping information for this record between these two systems can be removed (although other mappings for the deleted record may exist between a given device and other hosts for multi-way synchronization scenarios).

In scenarios without any deletion metadata, identifying deleted records may require a complex heuristic. Lacking any reliable metadata on which to base a decision, however, it may not be possible to correctly identify whether a record has been deleted on one system, or is simply missing on that system and should be copied from the other. This situation should be avoided whenever possible, as it can result in unreliable results.

Any record that has been identified as having been modified, but which hasnt been added or deleted, must by elimination be treated as updated. Updated records can be detected by comparing mapped records between the two devices and detecting differences. There are a wide variety of algorithms that can be used to compute a hash for a record on both devices [15]; this hash can then be compared, and if different

(assuming we use the same hashing criteria) can be marked as having been updated since the last synchronization. If one end of the synchronization maintains a modified record or a modification date and last synchronization date, we can determine which record (if not both) was updated since the last synchronization. Note however that if neither side maintains such information, determining which side contains the updated data will again be difficult: with a modified field that exists on both databases we will be unable to determine which version is the updated record and which was the original. Likewise, with a field that exists on one device but not the other, it becomes impossible to determine whether that field was added to one device (and thus requires update to the other), or if the other device has this field removed (in which case it should likewise be removed from the first device). Note that using a mechanism where we compare record hash values (or where we simply compare fields directly) is generally unsatisfactory in terms of synchronization time and the amount of data that needs to be transferred, as each set of records must be read and compared.

Once we are able to identify individual modified records, a sub-problem can be to communicate the set of modified records to the other end of the synchronization session in the fewest possible packets of data. In the situation where one (or both) devices are unable to determine for themselves which records have been modified since the last synchronization (i.e.: when the records of both devices need to be compared to determine if modifications have been made), such an optimization is difficult, and has generally not been undertaken. This sort of synchronization is often denoted as slow sync in the relevant literature [4, 6, 53, 12].

In the situation where both sides are capable of identifying which records have been modified independently, however, we can apply various protocols for communicating precisely which records have been modified since the last synchronization session. One algorithm (typically denoted as fast sync in the relevant literature [6, 51, 12]) involves a request-response style protocol, where one end of the synchro-

nization (the requester) sends a request for the next modified record in the relevant database. The responder responds either with the next modified record in an iteration of modified records, or a code denoting that no further modified records are available. This protocol thus requires only $n + 1$ packets (where n is the number of modified records in the database) to request all of the modified records in iteration in a one-way synchronization.

An interesting alternative method that has been proposed is Characteristic Polynomial Interpolation (CPI) [57, 6, 53]. CPI Sync has the significant benefit (as presented) of not requiring a system of modified flags nor modification date/time to determine which records on both hosts are new since the last synchronization. However, as the zero solutions to the interpolated function only give us those records that exist on one host but not the other, it is only ideal for the case where we want to detect new records that exist on one device, but not the other. While not presented in [53], given a mechanism for independently determining which records have been modified, we can run a second round of CPI Sync using the modified records on both devices as the subset. The resulting zero solutions of the interpolated function are thus the records that have been modified on one device, but not the other (as an interesting side effect, if we compare the solution values against the set of local records known to be modified, those local records which have been modified but which do not appear as a solution can instantly be flagged as having been modified on both devices, and hence potentially in conflict).

While [53] maintains that CPI Sync often requires significantly fewer packets than the Slow Sync algorithm, and discusses how the Fast Sync algorithm functions (and further compares Fast Sync performance to Slow Sync performance), it fails to compare the performance and number of packets required for synchronization to the Fast Sync algorithm. While this data is not discussed nor present in [53], we can derive some expectations. If we assume that we perform two rounds of CPI Sync to

determine both the added and modified records (as proposed above), for n modified records we would need to transmit at least $n+2$ packets for a one-way synchronization (one for each round of evaluations in CPI Sync, plus one to request each modified record). Without the second round of CPI Sync to determine the modified records, the entire algorithm becomes equivalent to that of the Slow Sync algorithm. As such, it appears that CPI Sync is best suited for applications where data is only ever added, and not modified, however even in this situation it does not appear to ever perform better than the Fast Sync algorithm.

2.8 Identification of Conflicting Records

Another important challenge occurs when we find that two records that map to each-other on both the mobile device and the synchronization host have both been modified since the last synchronization session. When this occurs, the records are said to be in conflict [41]. For the purposes of this paper, we identify the following conflict types between two systems device A and device B:

1. Two records with identical identification information was added to both A and B since the last synchronization,

2. A record was modified on either A or B, and was deleted on the other device since the last synchronization,

3. A record added to either device A or B, and was independently added and later deleted from the other device since the last synchronization,

4. A record was modified on both A and B since the last synchronization.

Note that the situation where a record was independently deleted from both A and B since the last synchronization is not typically considered a conflict, as deletion typically entails the same act on every database (i.e.: the record is simply no longer required). In essence, in such a situation the data is already synchronized for us

purely by circumstance, and the situation can be safely ignored (with any required final erasure of the records on both sides being performed as necessary).

Conflicts are often the most difficult situations to be resolved in data synchronization, and nearly always entail the use of a heuristic to resolve the conflict [45, 51, 56]. When this heuristic is imperfect, it may cause a situation where the merged data differs between the two devices after the synchronization session is complete, thus either forcing these records to be synchronized again during the next synchronization session, or appearing to the user as a failure in the synchronization process (as both devices now contain different data for the same record). Minimizing errors in the merge process to ensure that the resulting data is correct is thus key to user satisfaction in synchronization systems.

Complicating matters, user activity can cause merge conflicts that are difficult to detect and manage. For example, if a user accidentally deletes a record and decides to restore it manually (by re-entering the record information by hand), it is possible that they may modify (either purposefully or inadvertently) the data being entered in such a way that its mapping with other synchronized databases is lost (*e.g.*: in a system that identifies records by a computed field hash, if one of the hashed fields is misspelled, uses different type case, etc., the field in question will exhibit a different hash, and thus the system will determine it to be a completely different record). In such a case, where the user expects the system to realize that the re-created record is supposed to be the same information as was stored in the accidentally deleted record, the system might in such a case decide that it is a new record, and handle it as such.

Detecting these conflicts is thus of high importance in the synchronization process. The system for determining the set of conflicting records typically depends upon a comparison of the modified record metadata available on both devices. In the case where modified since last synchronization flags are in use, detecting a conflict simply requires finding the subset of all records where a given record has its modification

flag set on both the mobile device and the host system. In a system that uses a modification time and last synchronized date field, conflicting records are those which have been modified on both the mobile device and host system since the last synchronization time.

For systems which do not employ synchronization metadata, detecting conflicting records requires the comparison between all pairs of mapped records. In such a case, however, it is often difficult to determine whether a given pair of mapped records which exhibit differences are simply modified on one side and unmodified since the last synchronization on the other, or whether a conflict exists. For this reason, the merging algorithms on such systems are often complicated and error-prone [56].

It is worth noting that two identical records may appear to be in conflict. This typically occurs when the user inadvertently thwarts the synchronization process by manually modifying a mapper record pair on both the mobile and host system with identical data. In such situation, modification flags and/or times may indicate that a record is in conflict, however an examination of the data on both may show that they already contain the same record data. In such a situation, no action needs to be taken other than to update the synchronization metadata for the records.

2.9 Conflict Resolution

Once the set of all pairs of conflicting records has been determined (if any), the conflicts must be merged together to create a record which contains the details of both modifications. Merging is typically the most computationally intensive operation of a data synchronization system, and a variety of merging strategies exist [14, 51, 56, 31].

It must be noted that the merge strategy in use for a synchronization system is typically dependant on the data involved. Different data types will have different merge strategies depending on the type of conflict, and the way the data is to be recorded on the device.

In some situations, the data may be presented as a listing where a merge simply involves appending any new data to the end of the old data. This is particularly common for text-based data. Merging such information can be done in a similar fashion as with version control repository synchronization tools such as the Concurrent Versioning System (CVS) [48] or Subversion [17]. Note that in the case of the source code synchronization tools, merge conflicts can still occur at the line level, repair of which requires user attention [14, 17].

For complex records that contain a number of short, simple fields however, handling merge conflicts can be more troublesome. The simplest algorithm to deal with such conflicts is to permit the user to specify a global preference as to which side of the synchronization should take precedence in the event of a conflict. With such a setting available, when a conflict occurs the higher-precedence devices record field data overwrites the lower-precedence devices matching record field. It is also possible to simply ask the user on a record-by-record basis [8], however this is often undesirable in situations where the synchronization is to occur without user intervention, as is typically the case where large data-sets are involved.

In implementations where individual fields have a modification time stamp, it is possible for the merge heuristic to simply choose the most recently modified field as being appropriate. While this assumption of appropriateness is not guaranteed to be correct, it is often suitable for use in situations where data consistency is more important than guaranteed correctness, as inconsistencies can lead to a variety of cascading synchronization problems.

Next, assuming the applications to be synchronized are designed with it in mind, it may be possible to retain both sets of field data within one field, so that the user can choose which version of the field is correct after synchronization (typically the next time the record involved is accessed). Any modification to the field (i.e.: the user deleting the invalid version of the field data) will cause the record to be

synchronized during the next session, at which time the record will be repaired on the next synchronized device.

Finally, in certain very special cases it may be possible to feed differing numerical values into an algorithm to come up with a new value for the field being synchronized. While this sort of synchronization is of limited use, it is implemented in Cristian's clock synchronization algorithm [18] to bring two clocks into closer synchronization, by taking into account the round-trip time required to transmit the synchronization request and response, and adding this to the current time as reported by the server.

2.10 Determining what information to include in the update packet

In order to affect changes to a remote record once the differences have been determined and any conflicts have been resolved, the modified data needs to be sent across the communications line to the remote device. Typically when synchronizing semi-connected devices, the more capable system (typically the synchronization host) takes care of these more computationally intensive operations (due to its greater available RAM, faster processing speed, and larger secondary storage facilities). Once the system has determined the updates to be made on the remote device, it has to construct an update packet containing the new record information.

As we typically wish to minimize the time required to synchronize the devices, it is highly desirable to minimize the data to be transmitted. This is particularly so for semi-connected devices, as their network connectivity may at times incur a monetary cost from the amount of data transferred, the time taken to transfer the data, or both [25, 54]. As such, minimizing the size of the update packets (along with minimizing the number of update packets) is highly desirable.

There are a variety of mechanisms for reducing the amount of data required for an update packet. One relatively obvious method is to simply use a lossless com-

pression algorithm on the record data prior to transmitting it as the update packet to the remote device. This method should be able to achieve an improvement in the update packet size, however it does incur a time penalty for compression and decompression. This is of particular importance on the mobile device, which typically has significantly less computational power than a desktop or server machine. For databases containing thousands of records, the added computation required to generate or process an update packet may become noticeable to the end user on slow, low-power, or memory constrained devices. In addition, compression and decompression will be an added power drain on the device, increasing power use. Hardware compression and decompression in the network transport link layer can mitigate this disadvantage, however mobile devices often omit such hardware due to space, cost, and power constraints.

A second option is to prune down the data contained within the update packet. While many synchronization systems effectively merge the entire record on the host or server side, and transmit the new record data to the mobile device as a whole, it is possible to construct a synchronization system that provides only the modified and merged fields, along with a set of field identifiers [13]. The field identifiers are used to denote the specific fields being changed by the update packet, along with those changes. In this way, small changes to records with a significantly large number of fields can be transmitted in a potentially significantly quicker manner [13].

Further refinements may be found by expressing field modifications not as their absolute final state, but by the difference between the old state and the final, synchronized state [17, 27, 28]. The suitability of a field-level difference calculation to a synchronization solution may depend strongly upon how radically fields are changed. If changes are typically relatively small, then transmitting only the differences may make sense, however in cases where data that is modified does typically change radically, such a system may in fact increase the amount of data that needs to be

transmitted when used in conjunction with the elimination of unchanged fields from the update packet [28]. In cases where the typical change level is difficult (or impossible) to quantify, a flag could be used to differentiate between fields that have been expressed as a difference from their previous value, and a field which is presented as a wholesale change.

Ultimately, if the processing power of the handheld device doesnt present a significant issue, and if the data (and the type of typical modifications) permit it, a combination of all three mechanisms may prove best, whereby field modifications are expressed as differences (or not, depending on the level of field change), only those fields which have been modified are transmitted (along with field identifiers so the receiving device can effect the same changes or replace the field as appropriate), and this update packet compressed using a suitable lossless compression algorithm to minimize the data that needs to be transmitted, thus minimizing the time required to complete the synchronization process.

One special case is the update packet for deleting a record on the remote device. Such an update packet typically only requires the unique record identifier used by the remote device, along with a flag or command to denote that a deletion of the record needs to take place. Newly added record update packets, while treated the same as those for other types of update packets, may require that the remote device return the unique record identifier for the newly created record, if the transmitter and received do not use the same record identification scheme. In this case, the returned unique identifier is used to update the record mapping information for future synchronization sessions.

2.11 Writing Updated Records and Removing Deleted Records

Upon completion of generating the update packet(s), they must be transmitted to the remote device, which decodes them, determines the type of modification to take place, and affects the necessary modification to its own target database. The specific steps to be taken are dependant upon the design of and information contained within the update packet itself, along with the structure of the database on the device being updated. The data presented in the update packet may require modification or conversions to fit within the target database. All of this is highly dependant upon the database design and target application, and thus cannot be suitably covered here.

One special consideration to note is the disposition of deleted records towards the end of the synchronization session. For cases where deleted records maintain a record of their existence (so as to determine which records have been deleted at synchronization time), once the deletion has been synchronized with all expected synchronizing devices it may be permanently erased from the system [51]. This is typically most significant for the mobile device, which typically has less available data storage facilities than the host(s) or server(s) it synchronizes with.

Determining when to remove these deleted record stubs is important, particularly in a multi-node synchronization solution. In the case where each mobile device synchronizes with only one synchronization host (one-to-one synchronization), removal of the stubs on the mobile device can be achieved immediately after each synchronization session. However, in situations where the mobile device must synchronize against multiple hosts (one-to-many synchronization model), these deletion stubs may need to be retained. In the situation where a mobile device synchronizes against a fixed number of other devices in the synchronization cloud, these stubs should be retained until they have been synchronized against each of the assigned devices.

In situations where the number of devices the mobile device synchronizes against is highly variable (such as in a complete synchronization network), these stubs need to be maintained until they have propagated to every other node in the network. As a given mobile device may not synchronize against every other member of the synchronization cloud directly, and as those devices may have synchronized with other devices which have been synchronized against the mobile containing the original deletion stub prior to synchronizing against that mobile device itself, we cannot easily determine when the deletion has propagated to all devices in the cloud. As such, a heuristic may be required to prevent these stubs from consuming mobile device memory unnecessarily; in the simplest form, these stubs may only be retained for either a fixed amount of time [51], or for a fixed number of synchronizations (or for some hybrid of these two values), trusting that devices in the cloud it has not synchronized with directly will eventually get the deletion information through other devices it has synchronized with. Another possible heuristic is to keep the deletion stub until it has synchronized with at least one server device; again, the strategy takes advantage of the assumption that the server will eventually propagate the deletion to all other devices in the cloud.

Likewise, any stored record mappings to or from any given deleted records ID can likewise be expunged from the system, once they are no longer valid. Doing so helps keep such mapping data to a size linear to the total number of synchronized records in the database, keeping the search time against such mapping tables relatively stable over time.

2.12 Updating Synchronization Metadata and Disconnection

To ensure the coherence of future synchronization sessions, any synchronization metadata must be updated. For synchronization systems utilizing modified flags, the flags

associated with the cloud member being synchronized against must be reset, such that all records show as being unmodified (when compared to the current cloud member being synchronized against) since the last synchronization. This permits us to properly determine those records which have been modified since the current synchronization [6, 12].

For synchronization systems utilizing a last synchronized time stamp, the time at which the synchronization of each record was completed needs to be updated [14, 51]. As well, the last synchronization time needs to be recorded [38]. Choosing the time for last synchronization time is important, particularly in scenarios where there are more than two devices in the synchronization cloud. Setting the last synchronization time too late (such as at the end of the synchronization session) can cause data loss in future synchronization sessions; records modified on another mobile device between the beginning of the synchronization and the time when the last synchronization time is recorded may be missed on subsequent synchronizations (as they will appear to be mode prior to the last synchronization time), however recording the last synchronization time at the beginning of the synchronization session may cause issues if the synchronization fails to complete. For this reason, the best policy it to record the beginning of the synchronization session for a given database as the last synchronization time, but to only commit this information if the synchronization completes correctly.

At this point, assuming no further databases require synchronization between the two devices, they may disconnect. The disconnection may be initiated by one side of the synchronization session by sending a disconnect message to the other device, or one device may simply disconnect at the protocol layer, terminating the connection.

It must be noted here that as mobile devices are semi-connected, an inadvertent disconnection could occur at any time. Wireless connectivity could be easily lost by moving outside of the range of the wireless access point, or by moving into a dead

spot (caused by a wall, overpass, building, etc.). Batteries could also fail during the synchronization process, forcing a shutdown of the mobile device. For this reason, it is important to be able to distinguish between intentional and unintentional disconnections, and to have suitable strategies for when an unintentional disconnection does occur, to ensure correct behaviour during the next synchronization session.

2.13 Conclusion

Data synchronization is a complex process, with many significant points of breakage. A variety of synchronization algorithms are possible, some with greater potential for breakage and data loss than others, but where limitations may be in place due to a lack of control of the design of one of the databases involved. Designing a synchronization system is complicated, requiring a careful analysis of the type of data being synchronized, how the mobile devices are expected to connect to the data within the synchronization cloud, and how the data is to flow through all the devices in the synchronization cloud network in a timely manner. Security, authentication, and data access rights cannot be ignored in most complicated synchronization environments, and for certain data types, legal restrictions and requirements may necessitate a very thorough analysis of the data-flow, to ensure that security and confidentiality are strictly adhered to.

The lack of a dedicated connection to any sort of network, and the ability for that connection to be lost at any time is the paramount consideration when designing a synchronization solution. As these devices are designed to be mobile, synchronization services must expect that at any point in time during the synchronization process that the user could leave the wireless coverage area, and yet still desire to use the device and the data it contains. For this reason, a synchronization solution for a mobile device must be graceful in its handling of lost connections. Dealing with such

unknowns is a significant challenge for designers of mobile device synchronization solutions.

Regardless of these issues, in a world where mobile devices have increasing capacities and capabilities, but where available wireless connectivity cannot be simply taken for granted, the need for better synchronization solutions will only continue to increase, and thus further research and development continues to be warranted.

Chapter 3

Related Work

3.1 Introduction

Mobile device data synchronization is a relatively new field of research, with the majority conducted within the past fifteen years. However, while mobile devices have some specialized needs and requirements, the basics are direct consequences of earlier research into file and database replication, as is used by distributed file systems and database systems. In particular, as mobile devices are portable and semi-connected, research into optimistic replication techniques for file and database systems has been the primary focus.

Research into file and database replication appears to have followed the rise of computer networking, with much research invested into this area during the mid to late 1980's. Work in file and database replication schemes has fed into (and been fed by) theoretical research in the study of set reconciliation.

Also of note is a parallel area of research into clock synchronization. While the problems in clock synchronization are different than in other data synchronization systems, many of these systems rely on synchronized clocks with a reasonable accuracy in order to perform tasks in set reconciliation, and thus I consider research into clock synchronization to be pertinent to this research.

In the previous chapter (2), I described the current research into mobile device synchronization. In this chapter, I will survey some of the important research from the areas of set reconciliation, file and database replication, optimistic replication, and clock synchronization.

3.2 Set Reconciliation

The problem of determining what information needs to be synchronized between two data sets can be generalized mathematically. This generalized problem is known as *Set Reconciliation*. This generalized problem has been researched extensively in a variety of different settings, including the areas of Error-Correcting Codes, Communication Complexity Theory, and Set Representation [44].

An early example of an efficient algorithm that can be used for testing set membership is the Bloom Filter [15]. In a Bloom Filter, records are fed into k hash functions, to obtain k positions into an array of size m. Each of these k hash results, the kth value in the filter array is set to 1. Once a set of records has been fed into the filter, a record can be tested to see if it has membership in the set by feeding it into the k hash functions, obtaining k array positions, and testing the values within the array for each of these positions If any of the k array positions is 0, then the record cannot have membership in the set. Bloom Filters guarantee that no false negatives can be reported, however false positives are not only possible, but as the number of records added to the set increases, the probability of false positives increases. Due to this, Bloom Filters do not provide a good solution to the Set Reconciliation problem.

The problem of set reconciliation is modelled as computing the symmetric difference shown in equation 3.1 [44]. In essence, it is the union of the differences between the two sets, such that the solution is the union of the set of records in set A that are not in set B, and those in set B that are not in set A.

$$S_A \oplus S_B = (S_A - S_B) \cup (S_B - S_A) \tag{3.1}$$

A number of solutions to 3.1 involving the computation of characteristic polynomials have been proposed [44, 6, 53], and have been demonstrated to have near optimal communication complexity, permitting the solution to 3.1 to be computed with minimal network packet exchange. In this scheme, characteristic polynomials are derived for both sets, based on a hash of their records. We treat these sets as being part of a sufficiently large finite field, so as to constrain the size of the polynomial evaluations, and to ensure integer evaluations for divisions. Once the polynomials are derived, a set of agreed upon sample points are used to evaluate the characteristic polynomials. These evaluations are then interpolated by solving a generalized Vandermonde system of equations. The zeroes of the resulting polynomials are determined, which results in the set differences for both devices respectively.

Solutions to the set reconciliation problem have also been proposed using generalized error control codes. In this system, the set reconciliation problem becomes a variation of the graph colouring problem, known as second order colouring [29]. Current research into this solution is limited to a one-way reconciliation, wherein only a single message of communication is permitted to complete the reconciliation.

3.2.1 Difference Isolation

Early research into the generalized notion of set reconciliation often frequently dealt with the sub-problem of difference isolation. Given two sets of data, difference isolation algorithms attempt to generate a result that describes the differences between one set versus the other. Implementations are frequently one-way, describing data elements that were removed in one of the sets, and added to the other [27, 28]. Difference Isolation algorithms can be seen as a one-way synchronization mechanism, and are used in a variety of contexts to find the differences between two files. A particular

problem space that Difference Isolation algorithms are used in is that of Source Code Management systems, such as Subversion and CVS [17, 48], where source file versions are stored as a series of differences from the previously stored version (or conversely, where previous stored versions are maintained as a series of differences from the most current version). In this way, storage is conserved, and it becomes relatively simple to move from one file revision to the next, while maintaining local, unsynchronized differences.

3.3 File and Database Replication

Another significant relevant area of existing research is in the field of File and Database replication. Large scale data storage in file systems and databases were early important drivers for computer use within government and corporate institutions, and with the rise of computer networking, research began into mechanisms where information could be distributed and replicated between multiple systems. This was undertaken for a variety of reasons: improved locality of information, better use of expensive disks, delegation of data administration, disconnected operation, and fault tolerance being significant research drivers [33].

Research into file and database replication began out of the drive towards distributed systems in the 1970's. One early paper [7] proposed a "tolerable, as well as tolerant resource sharing environment" based on a single primary server and multiple backup servers, any of which can perform the "primary duties" of responding to queries and modifications. These changes are then immediately synchronized to provide n-fault tolerance.

Later efforts [38, 33, 56, 37, 16, 21, 31, 45] continue to enhance and expand the research into file and database replication, offering such improvements as lazy replication strategies [38], disconnected operation [33], managing update conflicts [56], exploiting application semantics to guarantee correctness and improve performance

[37], improved policies to enhance data freshness [16], and using a transformational approach to safely and generically synchronize data [45].

Much of this research into file and database replication has led to actual implementations, such as the CODA File System [33], the Bayou weakly connected storage system [56], the Bengal Database Replication System [21], and the IceCube reconciliation framework [31]. These research platforms have been used to demonstrate the feasibility of differing approaches to file and database replication, and provide important reference platforms for implementing the above mentioned concepts into real-world systems.

3.3.1 Source Code Management Systems

A specialized application of file system replication is that of Source Code Management. In a source code management system, multiple revisions of every tracked file are maintained using delta storage [50] to maintain the differences between adjacent revisions of the same file, by using difference isolation mechanisms as discussed above in 3.2.1. Users interact with the system via a series of updates and commits, whereby either the local repository is brought into sync with changes made to a remote repository since the last update session, or a remote repository is updated to reflect changes made to the local repository since the last commit session [48, 17]. Files can exist within multiple revision streams that are rooted to the main code base (typically called the *trunk* [48, 17]), known as *branches* [23], which can later be synchronized back to the trunk.

It is worth noting that the typical synchronization operations in a source code management system are one-way, requiring both a commit and an update to bring both a local and a remote repository into sync. Files that contain modifications during these one-way synchronizations within both the local and the remote repositories can have their modifications automatically resolved if they exist within different parts of

the file, however when they occur within the same region of a given file, a conflict occurs which the user must manually resolve [48, 17].

3.4 Optimistic Replication

Replication solutions developed thus far fall into two distinct camps: those which use a record locking mechanism during changes (*pessimistic replication* [14]), and those which perform no record locking, but which require user interactive or heuristic mechanisms for dealing with conflicts (*optimistic replication*). Traditionally, distributed applications have used the pessimistic replication model, however the pessimistic model works best in situations where latencies are low, and failures are uncommon [51]. They also scale poorly [61].

Optimistic replication systems require no locking, and are based on the assumption that conflicts will occur rarely, if at all, and that when they do occur, they will be relatively easy to manage and correct. This can improve availability, as data will continue to be available even in the face of a network partition. In addition, optimistic protocols can boost overall system performance [46]. The benefits of optimistic replication schemes come at a cost however: replicas can diverge, and conflicts between concurrent operations can occur, which must be managed. In some cases, conflict management cannot be handled automatically, and thus may require user intervention to solve (e.g., as discussed in 3.3.1).

A detailed survey of optimistic replication is undertaken in [51].

3.5 Clock Synchronization

Data synchronization mechanisms have also been used for ensuring that clocks on differing systems are within an acceptable bound. Clocks can be used to determine a *happened before* partial ordering [39], and in order to effectively compute this ordering between events in a distributed system, it is desirable for the clocks between those systems to be effectively synchronized.

Clock synchronization has a special concern not typically found in other synchronization and replication systems, in that the value being synchronized is time itself. Time is continually progressing, and as such in order to ensure that the end result of the synchronization is that the two clocks on either end of the communication channel are within an acceptable precision value of each other, the message delay involved in transmitting the time value must be taken into account [18]. Thus, if system A sends time value t_A to system B, with message delay m, the time that B should configure itself to use (t_B) would be calculated as shown in equation 3.2:

$$t_B = t_A + m \tag{3.2}$$

The key area of research into clock synchronization is determining the value of m in equation 3.2. In Cristian's algorithm [18], m is calculated by dividing the round trip time from request to response in half, and using this value as the message delay. More advanced algorithms, such as the one presented in [19], can better resist transient faults through self-stabilization.

Chapter 4

Simulation Environment

4.1 Introduction

The purpose of the simulation environment is to provide a consistent mechanism for evaluating the cost and efficiency of various synchronization protocols. It takes a series of configuration parameters as input which describe the characteristics of the environment we wish to simulate along with how long to run the simulation, and outputs a linear equation expressing the average cost per day for the synchronization algorithm in question.

In this chapter, we'll explore the environmental elements being simulated, the mathematical models used to drive the simulation, how the simulation logic is controlled, the protocol used to communicate with synchronization adaptors, along with a discussion of the functionality of the three synchronization protocols developed to compare the ETS protocols effectiveness against.

The simulation itself runs at a resolution of one-second intervals. We define a *tick* to be a simulated one second interval within the simulation.

4.2 Simulated Elements

The goal of the simulation environment is to evaluate the overall cost of a synchronization algorithm by following a model user through a series of days as they[1] synchronize their wireless mobile device through a programmed series of locations. Within each location there may be a variety of network types available, each with their own cost structure and data transfer rates. These elements will each be presented in this section.

4.2.1 Networks

As we are evaluating the synchronization costs related to wireless mobile devices, we require some form of network transport for the transmission of synchronization data and metadata within our simulation environment. These are expressed as a series of network objects, one per network which our sample user may intersect during the simulated day.

The networks each have three primary attributes inside the simulation: their transfer rate expressed as bytes per second (N_{rate}), the cost associated with each byte transferred (N_{cost}), and the probability per second that the connection can be lost ($P_{connectionlost}(N)$). Each of these elements is an important part of the simulation execution, as they permit the simulation to determine the duration of a series of network operations, their related monetary cost, and at which probability a connection may be lost during a synchronization session.

4.2.2 Locations

A location is a specific geo-physical region that the user may be within during their simulated day. A location is simply defined by zero or more networks which are accessible within that location. Note that a given network may exist within multiple

[1]For the purposes of this paper, *they* is used as a gender-neutral identifier for a single individual, as supported by references [2, 20, 42].

locations; for example, it is expected that generally a cellular network will be available in most locations within most urban areas. Thus, the relationship between locations and networks is an N:M relationship.

4.2.3 Locales

Locales build upon the physical locations by layering atop them temporal information; thus a locale is a location along with a time in which that location is entered. This is used to permit the simulation to change from one location to the next during the duration of the simulated day.

Along with the time in which a location is entered (L_{entry}), the locale also provides information on both the frequency of synchronization ($L_{syncfrequency}$), and the number of expected mobile database accesses ($L_{expectedaccesses}$). These are described internally as the expected number of synchronizations per second, and the expected number of accesses per second. Note that we typically do not expect a user to make multiple synchronization requests nor to have multiple database accesses within a a single second; as such these values are expected to be represented by small positive fractions.

Through the use of locale objects, we can permit the user to enter the same physical location at different times of day, with different mobile device usage patterns. For example, a user who telecommutes from home and who uses their mobile device frequently during the normal work day may have a high expected database accesses value during business hours, but would have an exceedingly low expected database accesses at night while asleep. In such a case the location is static, but the users usage patterns change. Thus, the relationship between locations and locales is 1:N. As the days are simulated from time 00:00:00 until time 23:59:59, the first defined locale is automatically entered at 00:00:00, and the last defined local persists until 23:59:59.

4.2.4 Records

A record is the smallest unit of data exchanged during the synchronization process, and represents a logical unit of compound information. For the purposes of the simulation, we do not actually model the record or its data, but instead simply maintain a record identification number, size, and revision number. The identification number (R_{ID}) is used as a matching mechanism between a record stored in the mobile database and a record stored within the server database. The size (R_{size}) is used for calculating how long it will take to transfer the record, as well as how much its transfer will cost. Finally, the revision number ($R_{revision}$) is used as a mechanism for comparing matching records within the mobile and server databases to determine differences between the records.

The revision number is of particular importance, as it is used in two different contexts within the simulation environment. Firstly, it is used during a mobile database access by the simulated user to assess whether to increment the intangible cost factor which relates to the use of out-of-date information; if the revision number of the mobile database record M_k is less than that of the matching record on the server database S_k, the intangible cost factor is incremented by the difference $S_k - M_k$. Secondly, it is used to provide the synchronization protocol implementation with a list of records requiring synchronization, returning the set of all records with identification number k where $S_k > M_k$.

While it is possible for the record size to be specified manually, the record size is typically assigned during the simulations set-up stage randomly based on a Normal distribution (4.3.2). In this way the records can be generated with varying sizes, with a pre-set mean and variance.

Records also have an implicit probability of access by the user of the mobile device. The simulation assumes that records are not evenly used, but that instead certain records are accessed more frequently than others. This is intended to model typical

real-world experience with databases. While this probability is not stored within the record itself, it is based on a records ID number and the total number of records within the database containing it. This will be discussed further in sub-section 4.2.5.

4.2.5 Databases

A database is a logical collection of records. For the sake of the simulation environment, databases are always of a fixed size. The system maintains two separate simulated databases, one for the server, and one for the mobile device. Ideally, for the intangible cost of synchronization to be zero, these two databases should be identical at all times. However, in the typical situation, it is expected that these two databases will in fact differ for the bulk of the simulation run [60].

The server database S maintains an expected number of modification arrivals per second and a maximum number of simultaneous modification arrivals. These modifications reflect changes made to the database by other database clients, exclusive of our simulated mobile device, but inclusive of all other mobile devices within the organization that synchronize against the same database. Using a Poisson Process (4.3.4), a series of probabilities are calculated for each possible arrival value from 0 to $S_{maxarrivals}$ inclusive, using the expected number of modifications per second. During every simulated second, a random value is selected and compared against this series of probabilities to determine how many records should be modified during the current simulation time. Once a number v between 0 and $S_{maxarrivals}$ has been selected based on the calculated probabilities, v records within the database are selected for modification. These v records are selected at random using a Normal distribution (4.3.2), such that the record with ID $S_{max}/2$ has the highest probability of selection, and the records with ID 0 and S_{max} have a very low probability of selection, with all other values having a probability falling upon a Normal curve. Once each record to be modified has been identified, its revision number $R_{revision}$ is incremented by one.

The mobile database M handles whether or not the user has viewed a record during each one second interval in the simulation, based upon the expected mobile database accesses value stored in the current locale object. During every tick, we choose a random number n and compare it to the expected mobile database access rate per second in $L_{expectedaccesses}$. If $L_{expectedaccesses} >= n$, an access has occurred. We then determine which record has been accessed during this interval in a manner identical to that of how we determine which record to modify in the server database S, using a Normal distribution. In this way the most-modfied records on the server are also the most frequently accessed records on the mobile database. Unlike the server database, however, mobile database accesses are limited to one per tick, and as such form a Bernoulli Process (4.3.3).

4.2.6 User

Tying the above simulated elements together is the user object (U). Only a single simulated user exists within a simulation run, and they persist throughout every simulated day. The user object maintains the list of locales available to the user, and manages the transition between one locale and another during the simulated day. As such, the user object maintains a 1:N relationship with the locale objects. The user object also contains the single instance of the mobile database, and as such has a 1:1 relationship with the mobile database object.

4.3 Models

Manipulation of the simulated elements is governed by a series of mathematical models, which dictate the behaviour of the simulation process by ensuring the simulated elements remain within realistic bounds. The models transform pseudo-random numbers into values within a defined range in order to simulate unknown information that falls within a known (or expected) set of bounds [24, 32]. This section details these

models as they are utilized within the simulation environment, along with the rationale for their selection.

4.3.1 Pseudo-Random Number Generation

A key ingredient to simulating discrete, random events is the reliable generation of random numeric values [24]. For simulation, it is generally desirable to generate values which are as close to uniformly distributed within the interval $(0, 1)$ as possible. As it is impossible to generate random values using purely deterministic techniques [24], we must choose a pseudo-random number generator (PRNG) algorithm which produces uniformly distributed numeric values, with an extremely long period between repetitions. Additionally, the pseudo-random number generation algorithm should produce random values very quickly, as an algorithm which requires a large number of instructions may cause our simulation to require significant runtime to arrive at a result, as a single simulation run may require tens of millions random numbers (indeed, based on experimentation a one year simulation can easily require in excess of 30 million random values). Thus, a balance between the appearance of randomness, long period, equidistribution, and computational efficiency is the ideal trait of a pseudo-random number generator for use in our simulator.

The PRNG selected for the simulation is the Mersenne Twister algorithm [43], as it meets all of these requirements. Its period before values start to repeat themselves has been proven to be $2^{19937} - 1$, has a very high order of equidistribution, and passes numerous statistical tests for randomness. It is also quite fast. In testing during PRNG evaluation for the simulator, the LGPL Mersenne Twist implementation used [36] was slightly more efficient than the Mac OS X built-in random number generators `rand` and `drand48`. The fact that the efficiency is not worse than the platform's random number generator, coupled with its improved equidistribution and significantly longer period make it an excellent choice for discrete event simulation [43].

4.3.2 Normal Distribution

Characterizing the types of data that can be synchronized in a generalized manner is nearly impossible; the types of information that can be synchronized are as varied as any other form of computer data. In addition, without studying a specific and widely adopted synchronized database, it is nearly impossible to generalize the access frequency of the records within the database.

Due to these factors, a normal distribution is used for calculating both the size of records, and to determine which records are accessed and/or modified by the client and server. In the case of the record sizes, a mean size in kilobytes is provided, and record size values are calculated based upon a normal distribution around this mean. By default, to create records within a reasonable size for a handheld device, a mean of 10 KB is used, with a variance of 5. This ensures that within two standard deviations of the mean, approximately 95% of records will be within the range $[5, 15]$ KB.

During the simulation run, both the server and the user are in frequent need to select a record for viewing or modification. As the individual records are assumed to be accessed at different frequencies, with some records being accessed more frequently than others, a normal distribution is also applied to their calculated access frequencies.. In this case, the mean is one-half the database size. A record is selected by calculating a random normally distributed value, converting it into the percent possibility of its selection in the cumulative density function, and then multiplying this percentage by the total number of records. In this manner, records around the mean will be selected more often, while records around the ends of the database are selected less frequently.

Generation of random, standard normally distributed values (with a mean of 0.0 and a standard deviation of 1.0 ($\varphi_{0,1}$)) is accomplished via the polar form of the Box-Muller transformation, as presented in [34]. This transformation generates a pair of random, normally distributed values from a single uniformly distributed random

value (generated by the Mersenne Twister algorithm [43, 36]). During each iteration of the random normal value method, a pair of values is calculated, with one cached. During a subsequent call to the method, the cached value is returned.

Generation of random, normally distributed values with other means (μ) and standard deviations (σ) is also supported. Calculation of these values φ_{μ,σ^2} is based upon a random, standard normally distributed value $\varphi_{0,1}$, which is then transformed using the following equation [24]:

$$\varphi_{\mu,\sigma^2} = \mu + \sigma\varphi_{0,1} \tag{4.1}$$

In certain situations, it is desirable to calculate the probability of a value within a range occurring within a normal distribution. To calculate this, an algorithm is used to compute the inverse normal cumulative distribution function. This algorithm is presented in [5].

4.3.3 Bernoulli Process

For testing whether or not a discrete event occurs at a given time t, a Bernoulli Process is used [32]. During simulation set-up, an expected number of events per hour (E_{hour}) is read from the configuration file, and divided to determine a probability-per-second for the event ($P(n)$), such that:

$$P_{bernoulli}(1) = \frac{E_{hour}}{3600} \tag{4.2}$$

Every simulated second, an event trial occurs by selecting a uniformly distributed random number from the PRNG, and is compared against $P_{bernoulli}(1)$. If the value is less than or equal to $P_{bernoulli}(1)$, the event occurs. If it is less than $P_{bernoulli}(1)$, the event does not occur. Note that due to this event determination mechanism, values of ($E_{hour} > 3600$) are not supported. However, this limitation is not an issue for the

events where the Bernouilli Process is used, as sub-second accuracy is not provided, and values in excess of 3600 would require an average of more than one event per second. In cases where multiple events per second is desirable, a Poisson Process (4.3.4) is used instead.

Bernoulli Process events are used for events which either occur or not during a one second interval. Typically this is used to simulate user access to a handheld database record, as it is assumed the user may only access one record at a time. Every locale has an independent (E_{hour}) for handheld record accesses.

4.3.4 Poisson Process

For events which may occur multiple times in a single tick, a Poisson Process is utilized [24, 32, 16]. During simulation set-up, an expected number of events per hour (E_{hour}) is provided, along with a maximum number of permitted events per interval (E_{max}). From these two values, a series of cumulative event probabilities is calculated, in the range of $[0, E_{max}]$. That is, the probability of exactly n events occurring during an interval with arrival rate r and for interval t is calculated as:

$$P_{poisson}(n) = \sum_{k=0}^{n} \frac{(rt)^k}{k!} e^{-rt} \tag{4.3}$$

Every simulated second, an Poisson trial occurs by selecting a uniformly distributed random number Z from the PRNG, and finding the smallest n such that:

$$P(n) > Z \tag{4.4}$$

The result is the number of events which occurred during the current second.

Poisson Process events are used to simulate those elements which can have a variable number of occurrences per interval. In particular, this type of event is used

when simulating modifications to the server database (as we assume this database has multiple clients, and thus may see multiple modifications simultaneously).

4.4 Controllers

Tying together the simulated elements and the mathematical models for the simulation are a set of simulation controllers which manage the execution of the simulation, modification of the data elements, application of the models, and the recording of the results.

4.4.1 Time Controller

An important aspect of the simulation is the flow of time. Time has a significant impact on the entire simulation execution; all of the event-based models are evaluated at fixed time intervals. In addition, locale switching is time-based. Thus, the proper simulation and use of these elements requires some form of time controller.

The time controller as implemented in the simulation environment is relatively simple. It uses a 1 second resolution counter, which runs each day from 0 to 86400 within a loop. Within the loop, two types of possible events are evaluated: *tick* events, and *alarm* events. Both are implemented using a registration and call-back system, along with a protocol that registered routines must implement to satisfy the call-back subsystem.

Tick events are used for simulated elements that require evaluation within regular, one-second intervals. These include all simulated elements using either a Bernoulli Process (4.3.3) or a Poisson Process (4.3.4), along with any elements that use their own distribution or process which must be evaluated at regular intervals that cannot be pre-computed. During every iteration of the tick loop within the time controller, every registered tick listener has its `activateTick` method called, passing with it the current tick value. Note that tick listeners are not guaranteed to be called in any specific order.

Alarm events are used for simulated elements which desire an event to occur at a specific, specified tick during the simulated day. There are used by elements which know in advance when they would like to be activated, such as locale switching. During every tick iteration, any alarms registered waiting for the current tick value have their `activateAlarm` methods called, in no particular order. In order to enhance the simulation run, all alarm listeners are stored within a a series of arrays, each of which is associated with a specific alarm time. These arrays are themselves referenced from within a hash hashing the tick time against the array of listeners for that time. Only those tick values which have an associated alarm as represented by an array, minimizing the memory requirements and minimizing the search time required to test if any alarms need to be activated for any given tick.

The time controller itself is only capable of executing a single simulated day. For multi-day simulations, the time controller's `run` method is executed in sequence once per desired simulated day by the simulation controller, as discussed in subsection 4.4.4.

4.4.2 Cost Controller

The entire purpose of the simulation environment is to generate a result; the value of this result is managed by and stored within the cost controller. The cost controller for the simulation keeps track of the cost of accessing out-of-date information on the handheld device, the cost of using network resources, as well as the total data transferred within the simulation session. These three values are stored as running totals during the duration of the simulation run, with the final results presented as the mean average result over the total number of simulated days.

Modification of the real cost value within the cost controller is typically managed by the Synchronization Controller (4.4.3), which handles the simulation of the synchronization session and calculates the cost to synchronize a given record through a specified network. Modification of the intangible cost of using out-of-date informa-

tion is handled by the mobile database, which intercepts read requests against the database, and calculates the distance of the handheld record's revision from that of the matching server record's revision, adding the result to the running total for the intangible cost within the cost controller.

The cost controller can output its results as a linear equation of the form:

$$y = mx + b \tag{4.5}$$

where y is the total cost, m is the intangible disappointment factor, x is the unknown intangible disappointment unit cost, and b is the real network access cost. Note that as this is a standard linear equation, m is also the slope of the linear equation. When analyzing the results, a lower slope m indicates less financial risk associated with using out-of-date data, and a lower constant b indicates a lower average network cost. Better synchronization algorithms must thus strive to decrease both elements.

4.4.3 Synchronization Controller

The synchronization controller handles tasks on behalf of the different synchronization protocols. It encapsulates those functions which are of common need to most of the synchronization adapters (4.5), in particular, recording the initiation of a synchronization session (2.3), reporting the set of differences between the server database and the handheld database (2.7), synchronizing a single record (2.11), and handling the disconnection (2.12). Other requirements for a proper synchronization solution are not implemented, in significant part because there is no actual record data within the simulation.

There are two important features of note within the record synchronization routine. Firstly, it manages and maintains the calculation of cost information associated with a given network. This ensures consistency when comparing the results from different protocols, as every protocol must synchronize each record through the syn-

chronization controller. Secondly, it permits us to simulate random disconnections from each network, based on the disconnection probability attached to each. This is calculated by evaluating how many seconds it is expected to take to transfer each record, calculating the probability of a disconnection within this time period, retrieving a value from the PRNG, and comparing it to the calculated probability. If the PRNG generated value is less than the probability, a disconnection occurs, the synchronization protocol is immediate informed, and no other record synchronizations are permitted until a new synchronization session is established.

The synchronization controller also calculates the duration required to synchronize each record transmitted during a synchronization session. As all synchronization occurs between ticks within the simulation, this value is used to ensure that another synchronization session cannot be started while and existing the previous session is logically in progress.

4.4.4 Simulation Controller

The simulation controller ties the entire simulation engine together by providing the facility to load the simulation parameters from a file, and the permitting a multi-day simulation to occur. It uses an instance of the time controller (4.4.1) to run multiple single-day simulations, up to a user-specified maximum number of days. Additionally, the simulation controller permits the automated execution of a batch of these multi-day simulations with each individual session within the batch having its own uniquely generated server and handheld databases.

As part of its duties, the synchronization controller resets the time controller for each new day, and calculates and maintains the values required to calculate the variance (σ) of the cost from day-to-day. The variance is calculated using an algorithm derived from formulas described by Knuth in [34], section 4.2.2, as formulas 15 and 16. This algorithm has been enhanced to work with linear equations instead of real-numbered values, so as to generate results appropriate to the simulators solutions.

The simulation controller supports a call-back mechanism, for alerting an implementing application when individual days complete. This is used in the simulation front-end to update a progress bar, alerting the user that work is being accomplished.

4.5 Comparative Protocols

In order to successfully evaluate the Expedient Trickle Sync algorithm, a series of algorithms have been implemented for the sake of comparison. Two algorithms are implemented as an experimental control: the null synchronizer which produces a worst-case cost scenario by never synchronizing any data, and a random data synchronizer. In addition, two algorithms based on existing industry-standard synchronization mechanisms, Fast Sync [6, 51, 12] and Slow Sync [4, 6, 53, 12]. All four algorithms are presented within this section.

4.5.1 Null Synchronizer

The Null Synchronizer algorithm is an algorithm which provides no synchronization capabilities, providing a worst-case scenario for the intangible cost, while simultaneously providing a best-case scenario for the real cost. Lacking any synchronization features, records quickly become out-of-date, with the situation getting worse as time progresses. However, with the lack of synchronization is also the lack of use of any network resources to accomplish the synchronization. As such, the Null Synchronizer exists to provide a worst-case control value; no synchronization solution should ever have a result with a slope as steep as that for the null synchronizer.

Lacking any synchronization facilities, the null synchronizer models the real-world situation where a given user never synchronizes their mobile device with an associated centralized server, but where the centralized server sees frequent modification to the same information used by the mobile device user.

4.5.2 Random Synchronizer

The ransom synchronization algorithm doesn't model any real-world synchronization algorithms. It has been shown to perform poorly [16] and is simply provided as a control value. It synchronizes at random based on an expected number of synchronizations per hour within each distinct locale (as do the Slow Synchronizer (4.5.3) and Fast Synchronizer (4.5.4) algorithms), however during synchronization the random synchronizer selects a uniformly distributed random number of records to synchronize (n) from within the range $[0..M_{size}]$, and then selects n records from the mobile database at random to synchronize. It completely ignores whether or not the records selected even require synchronization, and thus may partially synchronize data which does not require it. In areas where multiple networks are available, one is selected at random for use during the synchronization session. As such, the Random Synchronizer uses minimal information to control the synchronization process.

As the random variate n used is uniformly distributed, over a large number of synchronizations the expected average number of records synchronized per session should be $M_{size}/2$.

4.5.3 Slow Synchronizer

The slow synchronizer simulates a set of real-world synchronization protocols, such as the Slow Sync mode of Palm Computing's HotSync Protocol [4, 6, 53, 12]. It is designed to read every record from the handheld database during every synchronization, determine on the server-side if any differences occur, and if so update the record and send it back to the mobile device. The slow sync algorithm is discussed further in 2.7.

In the simulator implementation, the slow synchronizer algorithm transfers every record in the mobile database across the network, with every modified record transferred twice. This mimics the real network transmission needs of slow sync style

protocols, which transmit every record to the server, which then compares them against its database, and transmits back any records which have been modified.

The slow synchronizer algorithm synchronizes at random based on the expected number of synchronizations per hour within each distinct locale, as a Bernoulli Process 4.3.3. This mimics the fact that slow sync based handhelds typically require the user to initiate the synchronization session.

4.5.4 Fast Synchronizer

The fast synchronizer algorithm simulates another set of real-world algorithms, such as the Fast Sync mode of Palm Computing's HotSync Protocol [6, 51, 12], Apple's iSync [8], and others. It is the most frequently used synchronization algorithm. Like the slow synchronization algorithm, it synchronizes at random based on the expected number of synchronizations per hour within each distinct locale. However, unlike the slow synchronizer algorithm, the fast synchronization algorithm is able to determine which records require synchronization, and only transmits the records which do across the network. As such, the rate of synchronization should be identical to the slow synchronizer algorithm, however the actual data transferred should be significantly reduced, due to the fact that only a subset of the records are transmitted during each synchronization session. The fast synchronization algorithm is discussed further in 2.7.

4.6 Limitations

While the simulation environment is designed to closely detail the realities of mobile data synchronization, it is imperfect, facing a number of limitations. However, it is felt that these limitations do not have a significant impact on the simulation results. These limitations will be discusses herein.

Firstly, the simulation environment only evaluates a single mobile device synchronizing against a single server instance. The source of the normally distributed

updates to the server not caused by the synchronization process is undefined. As the records themselves lack any actual data, the simulator assumes that all record conflicts are resolved automatically. As such, while the simulator only simulates one mobile client with one server, the changes that occur on the server itself could themselves be viewed as having come from other mobile clients which have synchronized prior to each synchronization session initiated by the simulated mobile client. It is likewise possible to view some or all of these background server record modifications as having come from other database servers. It is felt that all of these views of how changes on the server are brought about are valid for this experiment, with the final results dependant on (and only demonstrably valid for) the server modification arrival model.

Additionally, all synchronization performed is unidirectional; in effect we're always updating the handheld device based on changes generated on the server, but never vice-versa. However, as a bidirectional synchronization can be composed of two unidirectional synchronizations in opposing directions, and as this limitation affects all the simulated synchronization algorithms equally, it is felt that the impact on the results is negligible. Indeed, assuming the server database's expected modifications per hour rate includes expected modifications from the simulated mobile device, the impact on the results will be virtually nil.

Another issue in the simulation environment is that the cost associated with manual conflict resolution is completely ignored. As the simulation environment doesn't simulate user-initiated changes to the mobile database, and as it contains no real data, no conflicts occur. It can be argued that for databases where conflict resolution can be completed in a completely automated manner, however, that the cost of conflict management will be virtually nil, and thus can be easily ignored.

The mobile users daily behaviour within the simulation is assumed to be completely static. While this is not realistic, it has been known for some time that mobile

users exhibit various degrees of regularity in their behaviour patterns [60], and thus this is not felt to be a significant issue.

In addition to the above, no attention is paid in the cost calculation to mechanisms to reduce the amount of information transmitted in an update packet; the simulation assumes that the entire modified record is transmitted to the out-of-date database for synchronization. While most existing synchronization algorithms do nothing to reduce this data size, the use of data compression and/or delta packets (as described in 2.10) can be used to equally reduce the size of packets for any of the synchronization algorithms used (with the notable exception of the null synchronizer, as it lacks any synchronization facilities). Thus, for the sake of comparison, data reduction methods can be safely ignored within the simulation.

Finally, it is very important to note that the results of the experiments conducted are dependent upon the distributions and models used. While the models used are felt to be intuitively reasonable, a lack of available research into mobile device use patterns, synchronization patterns, record size distributions, and modification arrival rates means that more exacting models simply do not exist at this time. While intuition may suggest the models selected above may be reasonable, research into similar models involving the Internet indicate that they do not in fact adequately model the real world, and that the results of research based on such models is thus suspect [59]. As such, I make no claims as to the applicability of the results derived from this simulator for real-world use scenarios that fall outside the confines of the models selected herein.

None of these limitations are felt to have a significant impact on the usefulness of the final results; the end results continue to permit differing algorithms to be accurately compared to each other within the confines of the model assumptions. Removing many of these limitations would significantly impact the performance of the simulation, without impacting upon the results in a significant or meaningful

way. Thus it is felt that these limitations have little real impact on the usefulness of the final results.

Chapter 5

Expedient Trickle Sync Algorithm

5.1 Introduction

The primary goal of my Expedient Trickle Sync (ETS) algorithm is to reduce the cost involved with data synchronization. It does so by considering two factors: the cost involved with physically transferring bits across a network, and the cost conceptually incurred when the user makes use of out-of-date data . These two factors are frequently diametrically opposed: the network cost can easily be reduced to zero by never synchronizing any information, however the cost involved with the use of out-of-date data increases significantly as the data becomes increasingly out-of-date. Conversely, we can synchronize at a high rate of frequency to minimize the probability of using out-of-date information, however this can be at the expense of using more network time and resources than is otherwise necessary. Thus, minimizing the cost of synchronization is a fine balance between ensuring the data the user is most likely to need is up-to-date, while minimizing the frequency and amount of time spent synchronizing [60].

The ETS algorithm attempts to minimize the cost of synchronization via four heuristic sub-algorithms: determining when to synchronize, determining which available network to synchronize with, determining which subset of records to synchronize, and lastly by prioritizing the synchronization order. While the intent of these four

sub-algorithms is to reduce the overall cost of synchronization, they are not designed to guarantee absolute minimal synchronization cost; additionally, as they are based in part on a statistical analysis of the user's past needs, sudden changes in a user's habits could potentially circumvent these algorithms. However, as these new user patterns take hold, the algorithm can learn from them in order to once again attempt to minimize the network cost; as such, the goal in cost reduction of the ETS protocol may not always be optimal when measured across short durations, an intentional part of its design is to find a local minima which provides a lower cost point than traditional synchronization algorithms using ad-hoc methods.

The ad-hoc approach used in my ETS algorithm was selected in part due to power considerations, including both real battery consumption, and processing power required to execute the algorithm. The calculations are relatively simple, and should execute quickly on even low-powered hardware. The actual power requirements of the ETS algorithm were not measured for this thesis however; this is left to future research 8.1.

5.2 Determining when to synchronize

Determining when the synchronize so as to minimize cost is an extremely complex task. At a one-second resolution, there are 2^{86400} possible sets of synchronization points. Determining which members of these sets have the minimal cost in a rigourous manner each day would be inordinately computationally intensive, and thus is completely infeasible. As such, a heuristic mechanism that will quickly reduce the cost of synchronization with minimal computation is desirable. Such a heuristic must be suitable for running on a mobile device, and thus cannot require significant computing power or memory.

To fulfill these requirements, an algorithm that we can very quickly evaluate is important. Easily calculated and recorded values, such as those pertaining to the

probability the user will need a record on the device within a specified interval of time, how long it has been since the device was last synchronized, and the cost of synchronizing on the least expensive network in the mobile devices current location are excellent candidates for creating such an algorithm.

The mechanism selected for the ETS algorithm to use is to combine the above mentioned values into an equation, such that high probabilities of use in the short term, coupled with increasing times since last synchronization have an increasing influence on the expression, and where high cost has a decreasing influence on the expression. The derived value is then compared against a threshold value (T): if the value is greater than the threshold at the time it is computed, we initiate a synchronization session, otherwise we do not.

In specific, given a value n representing the number of time intervals to look ahead into, the probability function $P(x)$ that generates the probability that the user will access any record on the mobile device during the fixed-size interval x, Δt is the time since the last synchronization session, and $N_{cost_{current}}$ is the cost of transferring one byte of data on the selected network, then we define the function G as:

$$G(x) = \frac{\left(\sum_{i=1}^{n} \frac{2^{n-i} \cdot P(x+i)}{2^n - 1} \right) \cdot \Delta t}{N_{cost_{current}}} \tag{5.1}$$

Note that in the function G we evaluate the mobile device record access probabilities of the next n time intervals, with each subsequent interval having half the influence of its immediately preceding interval. In the default implementation, the intervals are 15 minutes in length, and the next 16 intervals are considered; thus the probability calculation takes into consideration the next four hours worth of usage. The rationale for using such a long period of time is in order to ensure that the probability factor is significantly less likely to be equal to exactly 0, as when it does the numerator in function G will become zero, and even when within a virtually free

network we will fail to synchronize. By considering a lengthy span of time, if we are in a sufficiently inexpensive network we can synchronize even if the device won't be used for up to the next 4 hours. Simultaneously, we can better avoid a situation that could frequently occur with smaller values of n where we avoid synchronizing because the numerator of G is zero while in an inexpensive network, but a short time later while in an expensive network we are forced to synchronize because the time since the last synchronization is very high (due to the lack of synchronization in the previous intervals). This is particularly noticeable in user models where the device goes unused for lengthy periods of time (such as while the user is asleep at night), as with a small value of n we may avoid synchronizing all night, only to be forced to synchronize as soon as the probability of access becomes non-zero due to the large Δt factor. Note that if the summation does evaluate to zero (that is, there is zero probability of the user accessing a record within the next four hours), G will evaluate to zero. As the threshold value T cannot be zero, the ETS algorithm will only ever synchronize data if there is a non-zero probability of user access within the next four hour window.

Calculation of the interval probabilities is based upon a table maintained by the ETS algorithm, that stores the number of record accesses for each time interval in the day, for each of the last seven days. The quantity of records accessed within each interval is not considered, only whether or not *any* record was accessed during the interval; the probability algorithm is interested only in the probability of a record being accessed each day during the specified interval, and not how many were accessed during each interval. Thus, an interval x_a that has accesses in the seven preceding days of $\{1, 1, 1, 1, 1, 1, 1\}$ would have $P(x_a) = 1.0$, whereas an interval x_b that has access counts in the seven preceding days $\{7, 0, 0, 0, 0, 0, 0\}$ would have $P(x_b) = \frac{1}{7}$. Using only the last seven days of access information permits the system to adapt within a one week interval to changing record access habits by the user.

Given function 5.1, and given a function I that returns the fixed-sized time interval containing a specific instant of time t, we can determine whether or not to synchronize by evaluating the following inequality:

$$G(I(t)) > T \tag{5.2}$$

This inequality is evaluated every second during the day to determine whether or not synchronization needs to occur. Given a sufficiently low probability, a sufficiently low time since the last synchronization, a sufficiently high network transfer cost, or any combination of these three attributes, synchronization will not occur during the interval. Given a sufficiently high probability, long time since the last synchronization, or low network transfer cost, or any combination of the three, synchronization will occur.

It should be noted that synchronization when the network transfer cost is low presents certain potential problems. As the data transfer cost for a given network approaches zero, synchronization will always occur (assuming the probability summation is non-zero), and the time since the last synchronization (Δt) would always be equal to 1 second (as we would synchronize during every interval due to the minor cost impact synchronization would have in a "free network" environment). ETS prevents a divide-by-zero situation simply by avoiding it altogether – even a "free" network is assumed to have some minor cost for the electrical power to perform the transfer. A mobile device synchronizing during every 1s interval while in a "free" network zone is most likely impossible due to the time required to resolve the synchronization (which is assumed to be greater than one second in duration), and would be a significant battery drain. As such, the ETS algorithm prevents this situation from occurring by only permitting one synchronization during each fixed sized interval x so as to prevent an unreasonable use of resources by the data synchronization subsystem.

5.2.1 Evaluating the Threshold

The algorithm as described is heavily reliant upon the value of the threshold to influence the frequency of synchronization of the mobile devices data, and as such selection of the threshold value will have a direct impact on the cost of synchronization. As usage patterns and networks differ from user to user, it is impossible to pre-calculate the threshold value that specifically reduces cost; as such, a heuristic mechanism is used to "learn" an optimal value for the threshold that reduces overall cost of synchronization.

Learning the optimal threshold value is difficult, as both raising and lowering the threshold value from its current value has the potential to raise the cost. Figure 5.1 diagrams the threshold modification possibilities and how they can negatively impact the cost of either raising or lowering the threshold value.

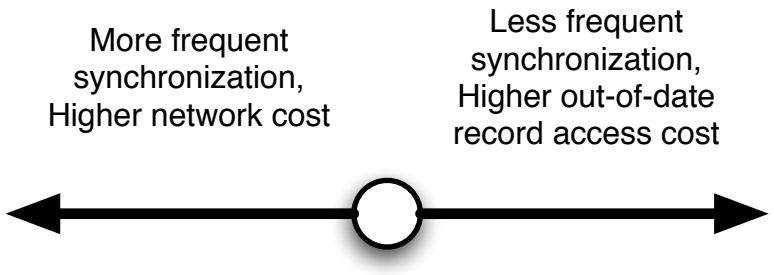

Figure 5.1: How modifying the threshold could negatively impact cost.

In order to determine whether cost improvements would occur when the threshold is increased, decreased, or by staying at the current threshold point, the ETS algorithm tests three different threshold points, within a range consisting of an *upper point*, a *lower point*, and *midpoint*. During intervening days, these three points are used for synchronization, and the cost of their use recorded. After analyzing the cost associated with each of these three points, one is selected to become the new midpoint, and new upper and lower bounds are computed.

The threshold evaluation subsystem is composed of two main algorithms: finding suitable initial threshold values, and computing new threshold values.

Threshold Initialization

In order to be effective, the threshold computation algorithm requires some reasonable initial values for the upper point, lower point, and midpoint. In particular, if all three initial values are too high, such that none will ever permit synchronization to occur, it becomes impossible to make any progress. Likewise, if the values are too low, we may synchronize during every interval for each of the values, again impeding progress. In both of these examples, the differences between the cost of synchronization for each becomes more dependent on random usage patterns, and we risk making a poor decision that will impact the cost in a negative manner. See subsection 5.2.2 for a more in-depth discussion of these and other potential threshold problems.

Selecting initial seed values for the threshold points is performed as a two-step process. In the first step, during the first day of device use, a fixed threshold is used, as presented in equation 5.3. It is felt that this value is a reasonable initial value, as it would permit a network with a per-byte transmission cost of $1 \cdot 10^{-12}$ to synchronize once every 800 seconds when the probability of use is 1.0. In addition, during the first day of use the device watches for and records the cost per byte of the most expensive network encountered.

$$T_{init} = 8 \cdot 10^{14} \tag{5.3}$$

Once the cost per byte of the most expensive network has been determined at the end of the first day, we can compute a reasonable threshold midpoint. The function used to compute the initial midpoint is presented in equation 5.4. This equation is a simplification of equation 5.1, where the probability summation is equal to 1.0, and

Δt is equal to one hour, expressed in seconds.

$$T_{midpoint_{init}} = \frac{3600}{max(N_{cost_{current}})} \tag{5.4}$$

Using this initial midpoint, we will only synchronize using the most expensive network if the probability summation times the interval since the last synchronization is equal to 3600, with the prototype case being where the probability summation is 1.0 and the interval since the last synchronization is one hour.

Next, the algorithm computes the initial upper bound point, using equation 5.5, that is the midpoint between a reduction of equation 5.1, where the probability summation is equal to 1.0 and Δt is equal to twelve hours (expressed in seconds), and the value of $T_{midpoint}$ from equation 5.4.

$$T_{upper_{init}} = \frac{\frac{43200}{max(N_{cost_{current}})} - T_{midpoint}}{2} \tag{5.5}$$

Finally, we compute the initial lower bound, using equation 5.6, that is simply the midpoint between $T_{midpoint}$ from equation 5.4, and a reduction of equation 5.1, where the probability summation is equal to 1.0 and Δt is equal to fifteen minutes (expressed in seconds), as presented in equation 5.6.

$$T_{lower_{init}} = \frac{T_{midpoint} - \frac{900}{max(N_{cost_{current}})}}{2} \tag{5.6}$$

The computation of these initial values helps to ensure that we seed the threshold evaluation algorithm with values where synchronization will occur at a reasonable frequency (that is, values that neither prevent synchronization from occurring, nor that cause synchronization to occur at every tick). Each of these three points will be evaluated by using them for synchronization on a subsequent day; after all three

points have been evaluated, they can be fed into the new threshold computation algorithm to come up with new threshold values based on their performance.

Computing New Threshold Values

While the initial threshold set should result in a system that synchronizes at various points through the user's day, it is neither expected nor assumed to be optimal. As such, after testing each set of thresholds points, their real-use cost is analyzed, a decision is made as to what the new threshold point set values should be, and the entire process is repeated.

Computing the threshold point cost for a given point T_a involves running the ETS algorithm for one day, with T_a as the threshold value. The cost equation for the day, as presented in equation 4.5 is computed, and is evaluated for $x = k$, where k is an estimated average cost per out-of-date record usage as input by the user. The resulting value is the *threshold point cost*.

To evaluate the threshold points, their costs are simply compared to each other. If T_{upper} is found to have resulted in the lowest cost, we increase $T_{midpoint}$. If $T_{midpoint}$ is found to have resulted in the lowest cost, we maintain $T_{midpoint}$ as is, and if T_{lower} is found to have resulted in the lowest cost, we decrease the value of $T_{midpoint}$. In the event that the cost of two or more of $\{T_{upper}, T_{midpoint}\}$ are found to be equal to T_{lower}, and that this cost value represents the lowest cost, we always decrease the value of $T_{midpoint}$. The rationale for this is, all other things being equal, that if we can synchronize more frequently for the same (or lower) cost, the end result will be a positive result for the user, as we will be able to communicate more information at an equal or lower cost.

The specific functions for increasing, decreasing, or maintaining the value of $T_{midpoint}$ are detailed in the following paragraphs. It should be noted that prior to modifying the threshold point set, the inequality in 5.7 is evaluated. If the inequality is true, no action is taken. This is done to prevent an eventual convergence

of T_{upper}, $T_{midpoint}$, and T_{lower} due to a large number of divisions against ever smaller differences between these values.

$$T_{upper} - T_{lower} > \epsilon \tag{5.7}$$

Increasing the Threshold If the value T_{upper} is found to be the least expensive option, we increase the current value of $T_{midpoint}$. This is accomplished by evaluating the set of equations 5.8, 5.9, and 5.10.

$$T_{lower_{new}} = T_{midpoint} \tag{5.8}$$

$$T_{upper_{new}} = T_{upper} \tag{5.9}$$

$$T_{midpoint_{new}} = \frac{T_{upper} - T_{midpoint}}{2} \tag{5.10}$$

Decreasing the Threshold If the value T_{lower} is found to be the least expensive option, we decrease the current value of $T_{midpoint}$. This is accomplished by evaluating the set of equations 5.11, 5.12, and 5.13.

$$T_{upper_{new}} = T_{midpoint} \tag{5.11}$$

$$T_{lower_{new}} = T_{lower} \tag{5.12}$$

$$T_{midpoint_{new}} = \frac{T_{midpoint} - T_{lower}}{2} \tag{5.13}$$

Retaining the Current Threshold If the value $T_{midpoint}$ is found to be the least expensive option, it is maintained as per equation 5.14. However, in order to prevent the case where we stop making any progress, we decrease T_{upper} by 10% of the difference of T_{upper} and T_{lower}, decrease T_{upper} by this value (equation 5.15), and increase T_{lower} by this same value (equation 5.16).

$$T_{midpoint_{new}} = T_{midpoint} \tag{5.14}$$

$$T_{upper_{new}} = T_{upper} - \frac{T_{upper} - T_{lower}}{10} \tag{5.15}$$

$$T_{lower_{new}} = T_{lower} + \frac{T_{upper} - T_{lower}}{10} \tag{5.16}$$

An example of this algorithm running is graphed in figure 5.2. After determining the initial point set ($x = 1$), we evaluate the points and determine that the upper point represents a cost decrease. A new point set is evaluated, resulting in the the points at $x = 2$. These points are evaluated, and the lower bound is found to represent the best cost, new points are evaluated, resulting in the points at $x = 3$. In the next iteration, the midpoint is selected as the least expensive, and so we maintain the existing midpoint and narrow the upper and lower bounds by 10%, resulting in the points at $x = 4$. Lastly, the upper point is found to have the lowest cost, and new points are once again evaluated, resulting in the point set at $x = 5$.

5.2.2 Problems with Thresholding

Thresholding can lead to a variety of potential issues that can negatively impact the overall algorithm. While care has been taken to design the algorithm in such a way as to minimize these issues, they are real possibilities, and a suitably crafted user pattern could certainly lead to problems. This subsection lists some of these issues, along with how the ETS algorithm attempts to mitigate or solve them.

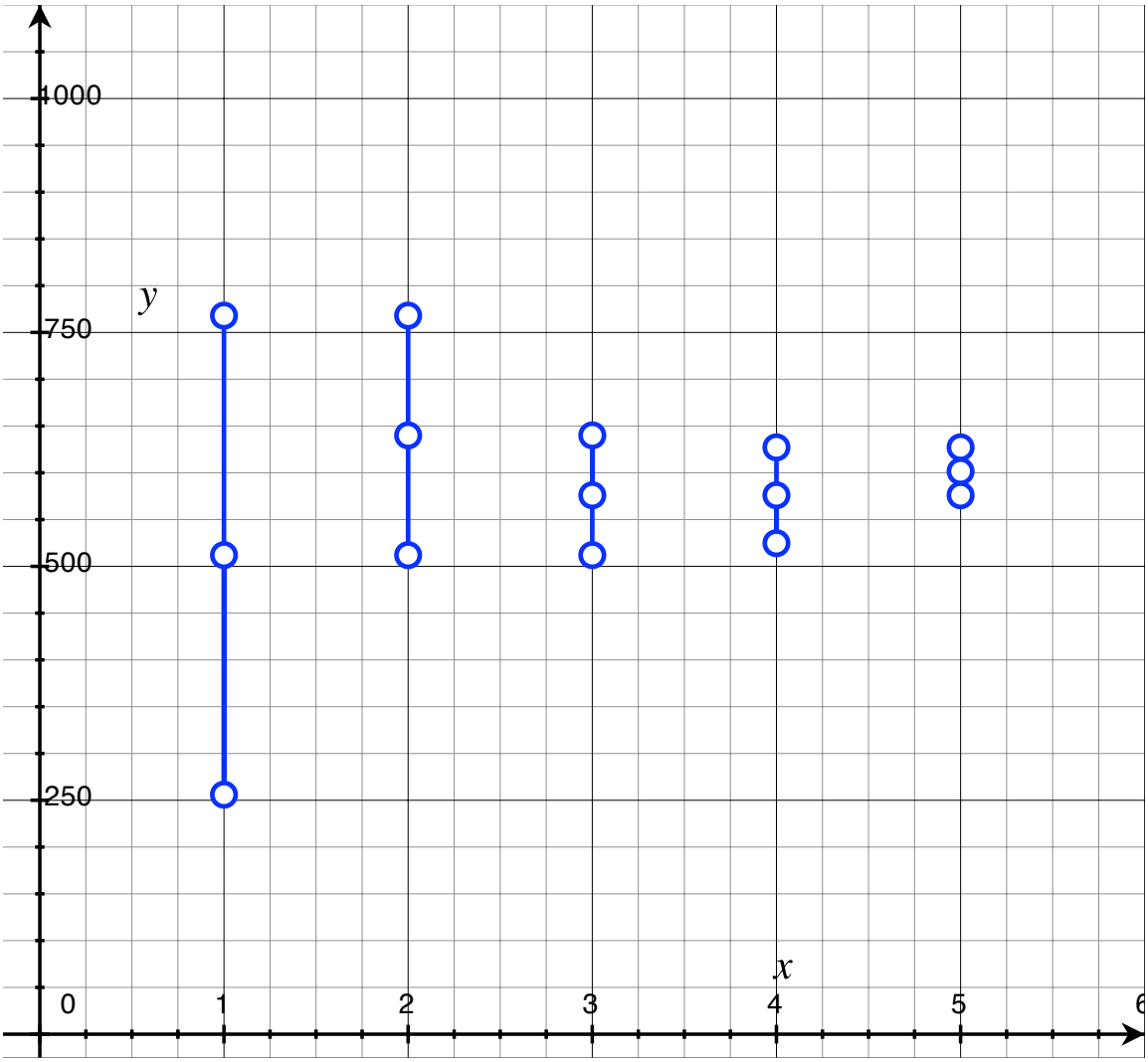

Figure 5.2: An example of a series of threshold evaluations. In the first evaluation, the upper point was found to be most cost effective, leading to the point set at $x = 2$. Next, the lower bound of this second set was selected, leading to the point set in $x = 3$. The midpoint at this third set was found least expensive, resulting in the point set at $x = 4$. Finally, the upper range in this fourth set was found to be least expensive, leading to the point set at x=5.

The threshold set points result in equal costs In the event that in evaluating T_{upper}, $T_{midpoint}$, and T_{lower} their associated costs are equal, it becomes difficult to make a decision as how to react. When this situation occurs, it may indicate that we are on a plateau, but with no way to evaluate this fact. This may not be a significant problem when the range between T_{lower} and T_{upper} is very small; indeed, this situation ideally eventually occurs, as it indicates we have found a local minima. However, if

the range is still large (bigger than ϵ), we have no way of knowing whether it is because the plateau is a suitable minima, or if it is in fact closer to a maxima. Figure 5.3 illustrates this problem.

If the range between T_{lower} and T_{upper} is large, and we are on a plateau, no progress will be made, as the midpoints between both T_{lower} and $T_{midpoint}$, and $T_{midpoint}$ and T_{upper} are equally poor candidates.

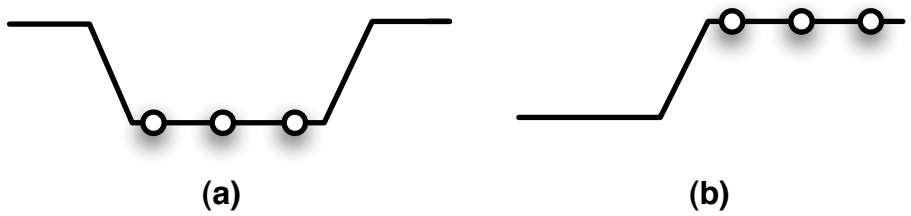

Figure 5.3: Two possible ways to have equal cost points without making progress. In situation (a), while it is impossible to make any progress, we are already at a local minima. In situation (b) however, further progress could be made, however given the knowledge of only the three points cost values, it is impossible to determine in which direction to progress. Additionally, it is impossible given just the three point values to determine if we are in situation (a) or (b). Note that in this graph the vertical axis indicates cost, while the horizontal axis indicates threshold values.

The current implementation of the ETS algorithm attempts to minimize this situation by computing reasonable initial threshold bounds. It is believed that situations as depicted in (b) in figure 5.3 only pose a significant problem at the two most extreme ends of the overall threshold range; that is, when the set of threshold evaluation points is close to 0, or when the set of threshold evaluation points are all outside the range where synchronization is possible (i.e., when there are no possible values for equation 5.1 that could cause $G(x)$ to surpass any of T_{lower}, $T_{midpoint}$, or T_{upper}). These two regions are problematic for either causing an unreasonably high synchronization frequency, or preventing any and all synchronization respectively.

In any case where the three threshold points are found to be equal, the ETS algorithm always selects the lower range as the new threshold range, as more frequent synchronization at no additional cost is always assumed to be a positive outcome.

Threshold Points are all out-of-range A special case of the previous issue occurs when the selected initial threshold set points are all of such a high value that it is impossible to ever meet any of the threshold points within a single day. In this case, the cost of using the algorithm becomes equal to the cost of simply never synchronizing at all, which is a highly undesirable trait. The ETS algorithm avoids this scenario by calculating reasonable initial values for T_{upper}, $T_{midpoint}$, and T_{lower} as documented above. Figure 5.4 illustrates this problem.

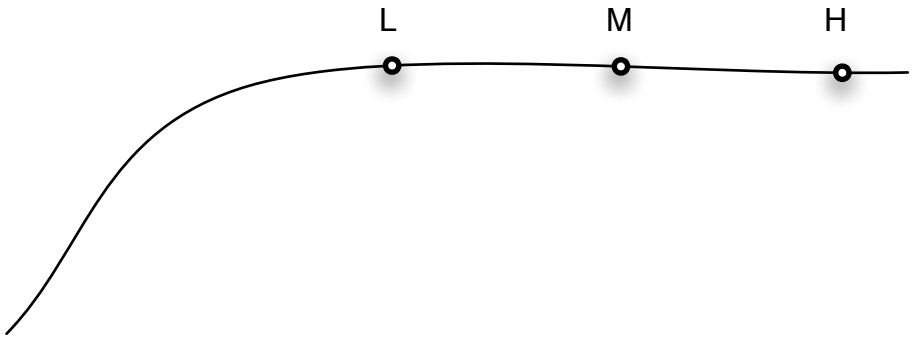

Figure 5.4: An example of all three points in the threshold set having equal cost. In this case, the lower, middle, and upper points all have equal cost, making is difficult to make a decision as to how to evaluate the next set of points, as all values within the range are equally poor. Note that in this graph the vertical axis indicates cost, while the horizontal axis indicates threshold values.

By ensuring that the initial maximum and minimum points within the threshold are within the synchronizing range (as described in 5.2.1 above), we ensure that all the threshold values are within a suitable initial use range, avoiding this problem altogether

Mid-point lands on a local maxima less than the high and low points If T_{upper} and T_{lower} are found to have a higher cost than $T_{midpoint}$, there is no guarantee that $T_{midpoint}$ is the lowest cost point within the current threshold range. Any point between T_{lower} and $T_{midpoint}$, or between $T_{midpoint}$ and T_{upper} may have a lower cost associated with it than $T_{midpoint}$ does. This situation is shown in figure 5.5. Given

a simple selection algorithm that selects $T_{midpoint}$ without modifying T_{upper} or T_{lower} would prevent us from finding a potentially more minimal point between these ranges.

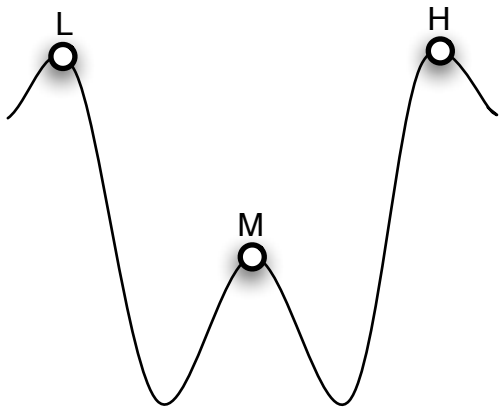

Figure 5.5: An example where the midpoint lands on a local maxima. In this case, while the midpoint is a better choice than either the upper or lower points, it would improve the cost profile by being either increased or decreased. Note that in this graph the vertical axis indicates cost, while the horizontal axis indicates threshold values.

The ETS algorithm removes this issue by ensuring that whenever $T_{midpoint}$ is selected as the minimal cost point, that T_{lower} is increased by 10% of the difference between T_{upper} and T_{lower}, while T_{upper} is decreased by this same amount, as described in equations 5.15 and 5.16. In this way, we continue to select $T_{midpoint}$ as the new threshold, but we decrease the range of the threshold set by 20%, causing us to potentially eventually find a new minima within the range in a future iteration of the threshold evaluation algorithm.

Atypical events can negatively impact cost for an otherwise good threshold point The entire cost evaluation for the ETS algorithm is governed by the assumption that the cost evaluated for a threshold point on any given day represents the typical cost for synchronization on that day. However, random events in the users behaviour, such as accessing more out-of-date records on one day rather than another can result in skewed cost results for a given day, which may result in the threshold selection algorithm selecting what could be, on average, a worse cost threshold point. Figure 5.6 illustrates this problem.

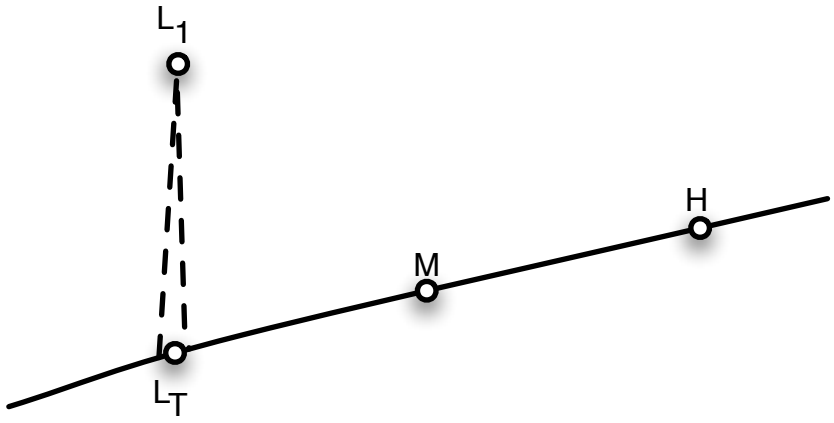

Figure 5.6: An example where a random use event causes a cost spike. In this case, when evaluating the cost of the lower bound, the cost for a specific day is L_1, whereas in the average case, it would otherwise by L_T. Because of this, the midpoint is selected as less expensive, leading us to a slightly higher cost region. Note that in this graph the vertical axis indicates cost, while the horizontal axis indicates threshold values.

Resolving this issue is difficult, as it is impossible to know at runtime whether a cost result for a given threshold point is the typical case or not. While it may be tempting to average the cost values by running the same threshold point for multiple days and then averaging the cost results, this solution can be very costly to the end user when we are testing what are very obviously poor threshold points multiple times. In addition, how many times we need to re-run the test in order to average out a single poor result is highly dependant upon the value of k as input into the threshold point cost function.

Possibly a better solution would be to evaluate each threshold point for j days, selecting the median cost value as the threshold point cost. This has the benefit of tending towards a more realistic value for each given threshold point, but once again has the disadvantage of bumping up the early cost of using the algorithm, as even obviously poor costing days will be run for j days, increasing their negative effect on overall cost of use.

In practice, it appears that the ETS algorithm's existing threshold management algorithms are able to sufficiently deal with this situation. While a potentially good

candidate may initially be avoided due to a random event causing a spike in the cost, the algorithm in the long run should tend back down towards this point if it does indeed indicate a cost benefit. In figure 5.6 for example, we will select the midpoint, reduce the threshold set magnitude by 20%, and assuming no other random spikes in the future, we will tend back down towards L_T as we "slide" down the slope of the graph. As such, mechanisms to better mitigate this problem are left to future research.

Not guaranteed to find the global minima It should be stressed that the ETS algorithm is not designed to find the threshold value that exhibits the absolute minimum cost. In practice, the range of threshold values that permit synchronization can easily surpass $1.0 \cdot 10^{308}$ (the order of magnitude for a 64-bit floating point number, as is used in the current implementation of the algorithm). Mathematically, as the cost per byte of transfer on the current network approaches 0, the value calculated by $G(x)$ will approach infinity, as expressed in equation 5.17.

$$\lim_{N_{cost_{current}} \to 0} \frac{\left(\sum_{i=1}^{n} \frac{2^{n-i} \cdot P(x+i)}{2^n - 1} \right) \cdot \Delta t}{N_{cost_{current}}} = \infty \tag{5.17}$$

Due to this fact, the ETS algorithm does not guarantee that it will find the absolute minimal cost threshold point, but instead attempts to find a suitable local minima through what is effectively a binary search of the overall threshold space. While this point may not be globally minimal, it is felt that the approach taken in the ETS algorithm both runs within a reasonable amount of time, and produces a result that will be generally better than existing initiation mechanisms, such as relying on user-initiated synchronization, or synchronizing every fixed unit of time.

Figure 5.7 demonstrates an example threshold value graph with both a local minimum and an absolute minimum. Because the local minimum traverses significantly more threshold point values than the global minimum point, the probability of the

ETS algorithm falling into this region as minimal is higher than for the global minimum, that only exists within a relatively narrow range of threshold points, and is surrounded by high-cost areas on either side.

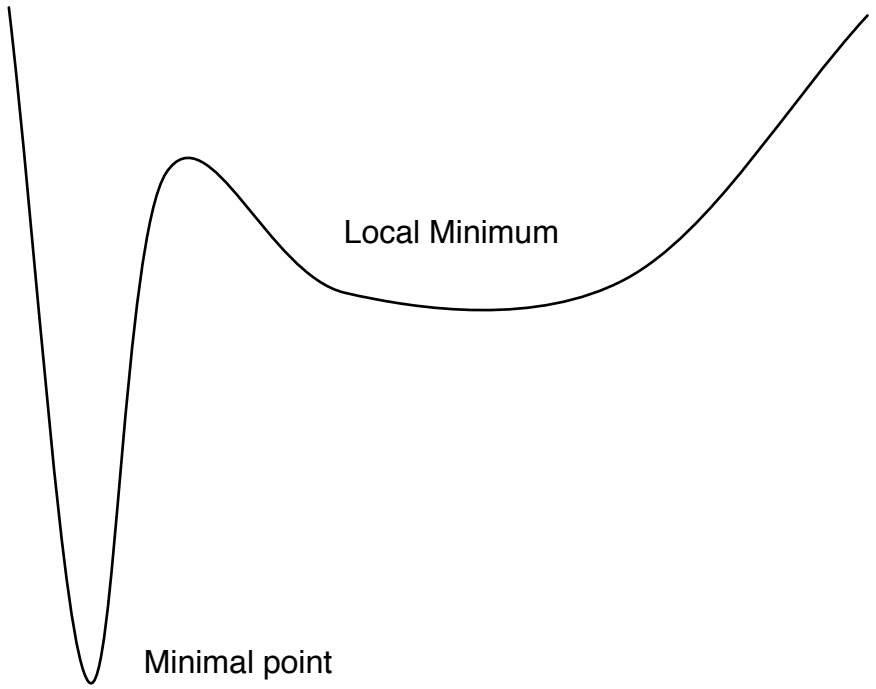

Figure 5.7: An example demonstrating narrow and broad local minima. In this situation, the algorithm is more likely to fall into the broad but shallow local minima, as we can get into it through more threshold points, whereas the global minimum has a sharper drop, making it easier for the algorithm to miss completely. Note that in this graph the vertical axis indicates cost, while the horizontal axis indicates threshold values.

5.3 Determining which network to synchronize with

Determining which network to use for synchronization is the simplest sub-algorithm involved in the ETS algorithm. When the mobile device is in the presence of multiple networks, and the decision has been made to synchronize, the network with the lowest cost per byte factor is selected for use during the synchronization session. If multiple networks with the same cost per byte are available, then the network with the fastest transfer rate at that cost is selected. If multiple networks with the same cost and transfer rate are present, one is selected at random. Real-world implementations of

the ETS protocol may decide to use further criteria for network selection (such as prioritizing those on a secure or trusted network).

5.4 Determining which records to synchronize

Once a synchronization session has been initiated, the ETS protocol must decide which records to synchronize. Unlike most existing data synchronization algorithms, the ETS algorithm does not always attempt to synchronize every modified record during the synchronization session, but instead uses a variety of metrics to balance the real cost of synchronizing each record versus the expected intangible cost of not having said record up-to-date.

Given the set of records needing synchronization Q, with size $|Q| = n$ and individual records $r_i \in Q$, where i is the records index value in the set of records to be synchronized, the function $C(R)$ that calculates the cost of transferring a record R via the selected network, the function P_{access} that calculates the probability that a record will be accessed by the user, the function $V(R, d)$ that calculates the version number of a given record R on a given device d (where s is the server and h is the mobile device), the constant k denoting the expected cost of using out-of-date information, and decision variables $x_i \in \{0, 1\}$, and a bar operator applicable to x_i which performs a binary NOT operation, the expected cost XC of the synchronization can be expressed as:

$$XC = \sum_{i=1}^{n} C(r_i)x_i + P_{access}(r_i)\overline{x_i} \cdot (V(r_i, s) - V(r_i, h)) \cdot k \tag{5.18}$$

Thus the goal of the record selection sub-algorithm is to minimize XP by solving for all x_i. This can be done by testing the following equality for each r_i requiring synchronization, and evaluating the corresponding $x_i = 1$ when it holds true:

$$C(r_i) < P(r_i) \cdot (V(r_{i_s}) - V(r_{i_h})) \cdot k \tag{5.19}$$

In essence, we are directly comparing the cost of synchronizing the records, versus the expected cost of not synchronizing the record. If the real network cost of synchronizing the record is higher than the cost of not having the record available, the ETS algorithm will not synchronize it during this session. If the real cost of synchronizing the record is lower than the cost of not having the record available, the record will be synchronized.

In practice, the above inequality implies that when we are using a sufficiently expensive network, not all records will necessarily be synchronized, but instead only those records that either have a high expectation of use by the user, that are sufficiently out-of-date, or a combination of these two factors. When using a sufficiently inexpensive network however, virtually all records will be expected to be synchronized.

The key to the proper functioning of this sub-algorithm is the suitable selection of a value for k. If k is set too high, the sub-algorithm will synchronize every record during every session, whereas if k is set too low, it is possible that records will never be synchronized. As the value of k is application and domain specific, it is impossible for the ETS algorithm to determine this value on its own, and thus must be obtained from an external source, such as the user or an administrator.

5.5 Prioritizing synchronization order

Finally, as its last step prior to exchanging records between the server and the mobile device, the ETS algorithm re-orders the records to be synchronized based on their probability P of access. This is done in an attempt to ensure that, should a network disconnection occur at synchronization time t, the records yet to be transferred will have a lower access probability than the records that have already been transferred (if any). This is performed as an aid in cost reduction when network disconnections are proportionately high; there is no cost savings involved with this sub-algorithm

when all data that has been selected for transfer during the current synchronization session is successfully exchanged.

5.6 Comments

We are confident that the ETS algorithm as presented represents a computationally effective way of reducing the overall cost factors involved with synchronizing handheld devices using an optimistic replication model. By using the simulation environment detailed in Chapter 4, Chapter 7 will detail the experimental results of the ETS algorithm, compared alongside the other algorithms described in Chapter 4.

In this chapter, we presented a number of graphs displaying possible computed threshold point costs versus threshold point values. As part of this conclusion, it should be noted that no such graphs have in fact been computed. The reason for this is demonstrated in part by by the result of equation 5.17; the range of values that require evaluation is massive; even evaluating only even n points for some reasonable value of n would be a significant undertaking. There was insufficient time or computational power available at the time this research was conducted to compute such a graph for a fixed day, however computing such a graph, given a sufficiently large distributed computing infrastructure and a modified version of the simulation engine as described in chapter 4 could generate such a representational graph. Generating such graphs for a variety of user models would be very useful in evaluating the overall effectiveness of the ETS algorithm, and would be useful in better determining the overall problem space the algorithm attempts to optimize. The generation of such graphs is thus left for, and is highly recommended as a future research topic in this field of study.

Chapter 6

User Model Study

6.1 Purpose

Accurate evaluation of the various synchronization algorithms within the simulation environment is dependant upon a set of accurate user models. These models dictate the frequency of record modification, data synchronization, and the users daily movements throughout their environment. An accurate model will thus provide a more useful result for each algorithm, and thus a more accurate cost comparison can be made.

Building accurate user models requires information on the habits and patterns of users with handheld mobile devices which are synchronized against a database. Unfortunately, such habits and patterns are lacking within the existing literature [40]. In addition, due to the fast evolution experienced in the mobile device arena, any possible existing studies older than one or two years may not reflect the habits of today's mobile device user. As such, a study of mobile device user habits was required in order to build a series of accurate, real-world user models to evaluate the simulated synchronization algorithms.

6.2 Methodology

In order to study the habits of the target population for the purpose of creating accurate user models, we created a series of seven survey questions. These questions are listed in Appendix E. The questions are designed to determine the average number of times users are expected to synchronize during both work hours and off-work hours, along with the size of the database being synchronized against and the average number of changes to said database during an average day. Four of the questions are designed to elicit specific numerical responses from which to design a user model, while the remaining three questions are designed to determine the respondent's confidence level for the values reported in three of the other four questions.

The survey itself is presented as a web application, using LimeSurvey [1]. This online survey tool was installed on a secure server at the University of Victoria, and was used to manage the survey invitations, provide the facility for the target population to complete the survey via the World-Wide Web, and to anonymize and store the survey results. Prior to completing the survey, the users are invited to download and read a Letter of Implied Consent, which is available in Appendix D.

Participants were self-selected, with invitations being sent to organizations known or believed to use data synchronization software. This was achieved by targeting Internet mailing lists whose purpose is the discussion of implementing mobile data synchronization solutions in multi-user organizations. Organizations known to use the jSyncManager [11] were also invited to participate. A copy of the invitation letter can be found in Appendix C. In order to minimize responses from organizations or individuals outside the target population, each invitation contained a survey-software generated token, which must be supplied in order to respond to the survey. Each token may only be used once, and no information is stored or retained to attach a specific invitee to a specific token.

The study was reviewed and approved by the University of Victoria Human Research Ethics Board.

6.3 Results

The survey was an abject failure, failing to garner a single response. The following reasons have been identified for the failure of this project, based on what few responses were received from the initial Letter of Invitation (see Appendix C):

Requested Information Confidential and/or Proprietary: At least one organization approached felt that the information requested is either confidential and/or proprietary in nature. While not indicating whether they had suitable information to respond to this survey or not, respondents in this category felt that this information on their synchronization habits may give them a competitive advantage, and thus were unwilling to share it, even when they were informed that only average/aggregate values were going to be requested, and that the survey would be completely anonymous.

Unable to Find Person with Required Knowledge: In virtually every case, finding the people responsible for handling what is typically a back-end process for most organizations is virtually impossible; publicly advertised contacts within organizations believed to use synchronization systems with mobile devices often didn't know whom to direct the survey to. Many requests appear to have been lost within organizational bureaucracy, never to be resolved. Invitations sent to public Internet mailing lists directed towards implementers of existing data synchronization solutions went unanswered; it is possible that those who are in charge of implementing such solutions do not have sufficient authority or interest in participating in such a survey.

No Response: The largest group of invitees fell into this category, having given no response. No response was received from any of the targeted mailing list

invitations to participate. The lack of response could be related to any of the above reasons, due to simple lack of interest, due to the feeling that participants wouldn't sufficiently benefit from the research, due to a lack of incentive, or because the selected target mailing lists lack sufficient members within the target audience.

6.4 Conclusion

While the lack of response to the survey was disappointing, it is fortunately not fatal to the research. Without suitable real-world information on data synchronization patterns, we must rely on invented models which appear plausible, in order to evaluate the synchronization algorithms previously described in 4.5. These invented models are presented in 7.2, along with arguments for their design.

The presence of real-world synchronization patterns would eliminate the possibility of bias in the selection of user model patterns for evaluating the ETS algorithm, and would continue to be of benefit to future research in this area. As such I would like to recommend that future research into the area of data synchronization for mobile devices should consider undertaking this sort of survey again, perhaps improving upon it through the participation of someone with more insider contact and knowledge within the mobile data synchronization implementation community, or through the use of some form of respondent incentive.

Chapter 7

Experiment and Results

7.1 Introduction

Evaluation of the ETS algorithm was performed by running it, along with the four other algorithms discussed in 4.5, through the Simulation environment described in Chapter 4, against a series of user models which will be detailed in 7.2. In this chapter, we will detail the procedure and results of these experiments.

7.2 Experimental Data

In order for the simulator to perform an experiment, we need to feed it two sets of information to control the simulation: one set to specify the parameters within the simulation that model the simulated user and their environment, and one set to control the duration and number of iterations for which the simulation should be run to generate meaningful results. In this section we denote the former set to be the *user model*, and the latter the *Experiment Parameters*. Each is discussed in the subsections below.

7.2.1 User Models

In order to perform an experiment, the simulation environment needs a user model to provide the set of elements to be simulated, and their parameters (as discussed in

section 4.2). Due to the failure of the User Model Study (discussed in Chapter 6), a set of user models was invented. This subsection details the parameters used for the six user models that were simulated to evaluate the different protocols.

Common amongst the six user models detailed below were the values selected the represent the different wireless network types users are typically expected to intersect with during an average day, namely "free" WiFi access zones (such as are frequently available at home, at work, etc.), commercial WiFi access zones, and GPRS-based cellular networks. The parameters for each are detailed in Table 7.1.

Network	**Transfer Rate (KiB/s)**	**Cost ($/B)**	**Disconnect Rate (disconnects/h)**
WiFi	49000.0	0.000000000001	0.01
Commercial WiFi	49000.0	0.000001	0.01
GPRS	57.0	0.000048828125	0.1

Table 7.1: Common network properties used in the evaluated User Models

The GPRS cost in Table 7.1 is based on a $0.03 per kilobyte transfer charge. The Commercial WiFi cost is based on a $10 per hour rate. The cost selected for WiFi connections is small but non-zero to accommodate for the cost of electricity and high speed network service.

Simple Users A and B

The first user model used to evaluate the simulated synchronization algorithms is a basic model entitled *Simple User A*, where the user remains in one location for the entire 24 hours day, with only WiFi based networking available. They have an expected synchronization rate of 4 synchronizations per day, with an expected number of mobile device record accesses set to five per hour. Simple User A synchronizes against a database that has 5000 records in it, with an average of 50 record updates per hour. For the purposes of the ETS algorithm, the estimated cost of using out-

of-date information is estimated to be $5. The XML source defining this user is available in listing F.1.

Simple User B is also the target of a further experiment, involving running the simulation for each of the estimated average out-of-date record access costs k, where $k = 1, 10, 100, 1000$. This was done to evaluate how the ETS algorithm reacts as the cost of out-of-date record accesses increases.

Location	Entry Time (s)	Expected Syncs/h	Expected Accesses/h
Home	0	0.166666666666667	5

Table 7.2: Basic Properties of Simple User A

Location	Networks
Home	WiFi

Table 7.3: Networks available to Simple User A by location

Records	Arrival Rate (per hour)	Maximum Arrivals (per interval)
5000	50	10

Table 7.4: Database properties for Simple User A

Simple User B is identical to Simple User A, however Simple User B only has access to a GPRS-based network. The XML source defining this user is available in listing F.2.

The purpose of these two users is to establish using a simple baseline how each synchronization algorithm compares when used in an all-day, single network environment.

Business User A, B, and C

The next model used to evaluate the synchronization algorithms is a more complex model, entitled *Business User A*. In this model, the user pursues a typical 09:00

Location	Networks
Home	GPRS

Table 7.5: Networks available to Simple User B by location

- 17:00 work day, with a one hour lunch break, and a one hour commute between home and work. Business User A uses a device which requires manual initiation of the synchronization process, and generally follows a corporate guideline which states that all users must synchronize once in the morning, and once in the afternoon. The basic parameters for this model user are detailed in table 7.6, with the networks available in each location listed in table 7.7, and the database properties in table 7.8. For the purposes of the ETS algorithm, the estimated cost of using out-of-date information is estimated to be $5. The XML source defining this user is available in listing F.3.

Location	Entry Time (s)	Expected Syncs/h	Expected Accesses/h
Home	0	0.0	0.00001
Home	25200	0.1	0.2
Car	28800	0.0	0.00001
Office	32400	0.25	10.0
Lunch	43200	0.1	2.0
Office	46800	0.25	10.0
Car	61200	0.0	0.00001
Home	64800	0.1	0.5
Home	79200	0.0	0.00001

Table 7.6: Basic Properties of Business User A

Location	Networks
Home	GPRS, WiFi
Car	GPRS
Office	GPRS, WiFi
Lunch	GPRS, Commercial WiFi

Table 7.7: Networks available to Business User A by location

Records	Arrival Rate (per hour)	Maximum Arrivals (per interval)
5000	40	10

Table 7.8: Database properties for Business User A

Business User B is identical to Business User A, except that Business User B uses a mobile device which can synchronize automatically once per hour [40], using the best available network (by cost). Otherwise, their day is identical, and they intersect with the same networks. As these modified properties have no effect on the ETS algorithm, which determines its own synchronization times, we can compare the results for the other algorithms against the results from the ETS algorithm run from Business User A. The properties for Business User B are presented in table 7.9. The XML source defining this user is available in listing F.4.

Location	Entry Time (s)	Expected Syncs/h	Expected Accesses/h
Home	0	1.0	0.00001
Home	25200	1.0	0.2
Car	28800	1.0	0.00001
Office	32400	1.0	10.0
Lunch	43200	1.0	2.0
Office	46800	1.0	10.0
Car	61200	1.0	0.00001
Home	64800	1.0	0.5
Home	79200	1.0	0.00001

Table 7.9: Basic Properties of Business User B

Business User C is a special experiment used to explore some of the issues encountered when synchronizing significantly larger data records, and how each synchronization algorithm performs under the stress of a much larger data set. In regard to the configuration of the simulation environment, Business User C is identical to Business User A, as in table 7.6. Modified for this experiment are the pre-defined record size limits built into the simulation environment itself, as discussed in 4.3.2. In

the Business User C experiment, the mean record size is increased to 4096 KB, with a deviation of 1024 KB. This size range mimics the observed average sizes of digital music files of the sort that may be synchronized by a mobile music player device.

Mobile Salesperson A

Mobile Salesperson A is a user that spends their typical workday on the road, visiting customer sites, some of whom permit the salesperson assess to a virtually free high-speed public WiFi network. The user spends significant time commuting between customer locations and their own companies office. Mobile Salesperson A owns a device which automatically synchronizes information once every 15 minutes when powered on, and which is powered off at night. For the purposes of the ETS algorithm, the estimated cost of using out-of-date information is estimated to be $10. The basic parameters for this model user are detailed in table 7.10, with the networks available in each location listed in table 7.11, and the database properties in table 7.12. The XML source defining this user is available in listing F.5.

Akin to Simple User B, Mobile Salesperson A is also the target of a further experiment into how the ETS algorithm reacts as the cost of out-of-date record accesses increases. This involves running the simulation for each of the estimated average out-of-date record access costs k, where $k = 1, 10, 100, 1000$.

Truck Driver A

Truck Driver A presents a special example of a semi-connected mobile device, in that instead of a handheld unit, a delivery truck contains a computer with an external antenna for connecting to different wireless networks. As such, the truck is the semi-connected, mobile device. In this model, the truck is of the type that does local deliveries during the day, and is typically driven through four locations: the depot where it is parked at night, on the road doing deliveries throughout the morning, parked at a truck stop at lunch, back on the road for the afternoon, and back to

Location	Entry Time (s)	Expected Syncs/h	Expected Accesses/h
Home	0	0.0	0.00001
Home	21600	4	4
Car	22500	4	5
office	24900	4	3
Customer Site	27000	4	2
Car	27600	4	5
Customer Site	27900	4	2
Car	29700	4	5
Customer Site	30000	4	2
Car	32400	4	5
Home	34200	4	4
Car	34800	4	5
Customer Site	35700	4	2
Car	37800	4	5
Customer Site	37920	4	2
Car	41400	4	5
office	44100	4	3
Car	46800	4	5
Customer Site	48600	4	2
Car	49500	4	5
Customer Site	49620	4	2
Car	50700	4	5
Customer Site	50820	4	2
Car	51900	4	5
Home	53100	4	5
Home	63000	4	5
Home	82800	0.0	0.00001

Table 7.10: Basic Properties of Mobile Salesperson A

Location	Networks
Home	GPRS, WiFi
Car	GPRS
Office	GPRS, WiFi
Customer Site	GPRS

Table 7.11: Networks available to Mobile Salesperson A by location

Records	Arrival Rate (per hour)	Maximum Arrivals (per interval)
1000	20	10

Table 7.12: Database properties for Mobile Salesperson A

the depot at the end of the work day. In this scenario, the truck is configured to automatically synchronize on average once per hour. The basic parameters for this model user are detailed in table 7.13, with the networks available in each location listed in table 7.14, and the database properties in table 7.15. For the purposes of the ETS algorithm, the estimated cost of using out-of-date information is estimated to be $250. The XML source defining this user is available in listing F.6.

Location	Entry Time (s)	Expected Syncs/h	Expected Accesses/h
Depot	0	1.0	0.00001
Road	32400	1.0	10.0
Truck Stop	43200	1.0	1.0
Road	46800	1.0	10.0
Office	61200	1.0	0.00001

Table 7.13: Basic Properties of Truck Driver A

Location	Networks
Depot	GPRS, WiFi
Road	GPRS
Truck Stop	GPRS, Commercial WiFi

Table 7.14: Networks available to Truck Driver A by location

7.2.2 Experiment Parameters

Each user model is run through a single simulated day for an entire year (365 days). After each simulated year, the simulation is automatically re-initialized and re-run. This process of re-initialization and re-running occurs 250 times to account for differences in the generated databases, and all other random variables which are generated

Records	Arrival Rate (per hour)	Maximum Arrivals (per interval)
2000	20	5

Table 7.15: Database properties for Truck Driver A

by the simulation at initialization time and which remain static for the duration of the simulated year.

7.3 Results

The simulation maintains track of the cost of accessing out-of-date information on the handheld device, the cost of using network resources, as well as the total data transferred within the simulation session, as detailed in section 4.4.2. The final average and variance are calculated on a day-to-day basis; that is the elements tracked within the Cost Controller are averaged at the end of the total simulation run by dividing the total accumulated cost and average data transferred values by 91250 (365 days per simulation run multiplied by 250 simulation runs), generating the final averaged results for the run. The variance for the run is automatically tracked, as it is not reset between simulation sessions, using the algorithm described by Knuth in [34] (section 4.2.2, formulas 15 and 16), as mentioned in 4.4.4. Because of this, the variance values reflect the variance on a day-to-day basis, and not the variance between simulation iterations. We present the results of running each of the above described user models through the simulation environment for each of the synchronization protocols tested below.

Note that for all the result sets below, while results for all five algorithms were computed for each user model, only the results for Slow Sync, Fast Sync, and the ETS algorithms are graphed. This is due to the fact that both Null Sync and Random Sync had such a high cost associated with them in every model that it was impossible to include them in a graph of reasonable size and resolution in this paper that could

include both, while at the same time allowing us to visually differentiate between the other three algorithms. As such, the ETS algorithm is graphically compared only to the Fast Sync and Slow Sync algorithms. However, the raw results for all five algorithms for each of the six user models evaluated are detailed in Appendix G.

In addition to the experiments comparing the different synchronization algorithms for each user model, an additional experiment was run for each model demonstrating the cost of synchronization using the ETS algorithm for each of 100 days. The purposes of these experiments is to demonstrate whether or not the ETS algorithm is indeed able to minimize the synchronization cost over time, as is intended. The results of these experiments are graphed, and are included below.

7.3.1 Simple User A

Simple User A demonstrates how the ETS algorithm can improve upon infrequent synchronization throughout the day, while maintaining a low network cost profile. In this model, as the only available network is virtually free the cost of synchronization is quite low. As the ETS algorithm can automatically determine for itself how frequently to synchronize, it can reduce the probability of using out-of-date record information in this user model. At the ETS target out-of-date record access cost of $5, both the Fast Sync and Slow Sync algorithms, synchronizing on average four times per day, would cost this user an average of $36.08, whereas the ETS algorithm would cost on average approximately $9.14. Figure 7.1 displays the results for this experiment. See table G.1 for the full results for Simple User A.

A sample daily cost for a 100 day run of the ETS algorithm for Simple User A can be seen in figure 7.2. Looking at the graph, we can see that in the first few days, a variety of high-cost options are evaluated, leading the algorithm to lower-cost opportunities on subsequent days. However, there are aberrations from time to time, with random spikes occurring at several intervals. These spikes would appear to be due to the use of records with a low access probability, which the ETS algorithm has

not synchronized for some time. These appear to have a significant impact on cost, and may indicate that these records need to be more aggressively synchronized as they progressively age.

7.3.2 Simple User B

Simple User B demonstrates the ETS algorithms ability to make a trade off between ensuring that the handhelds information is kept up-to-date, and the cost of synchronization. Looking at the Slow and Fast Sync algorithm results for Simple User B and comparing them with Simple User A, we can see that all four exhibit approximately the same slope. This is to be expected, as all four synchronize at exactly the same rate. Looking at their network access costs, we can see that for the Fast Sync and Slow Sync algorithms that these cost elements are roughly in proportion to the increase in cost between the WiFi and the GPRS networks.

Looking at the ETS algorithm, however, we can see that due to the added network cost, the slope for Simple User B has increased, reflecting an increase in average out-of-date record accesses. This increase reflects a decreased synchronization rate, in an attempt to decrease the network access cost element.

Simple User B also demonstrates that while the ETS algorithm does strive to minimize the overall costs related to data synchronization, it isn't necessarily able to find a global minimum cost. The the $5 average out-of-date record access cost used in the Simple User B user model, the Slow Sync algorithm would cost an average of $10540.38 per day, the Fast Sync algorithm would cost an average of $604.17 per day, while the ETS algorithm would cost an average of $607.74 per day. The ETS algorithm thus exhibits a nearly identical outcome when compared to Fast Sync at the specified target out-of-date access cost, however the ETS algorithm has the benefit of maintaining on average 41.6% fewer out-of-date record accesses.

Figure 7.3 displays the results for this experiment. See table G.2 for the full results for Simple User B.

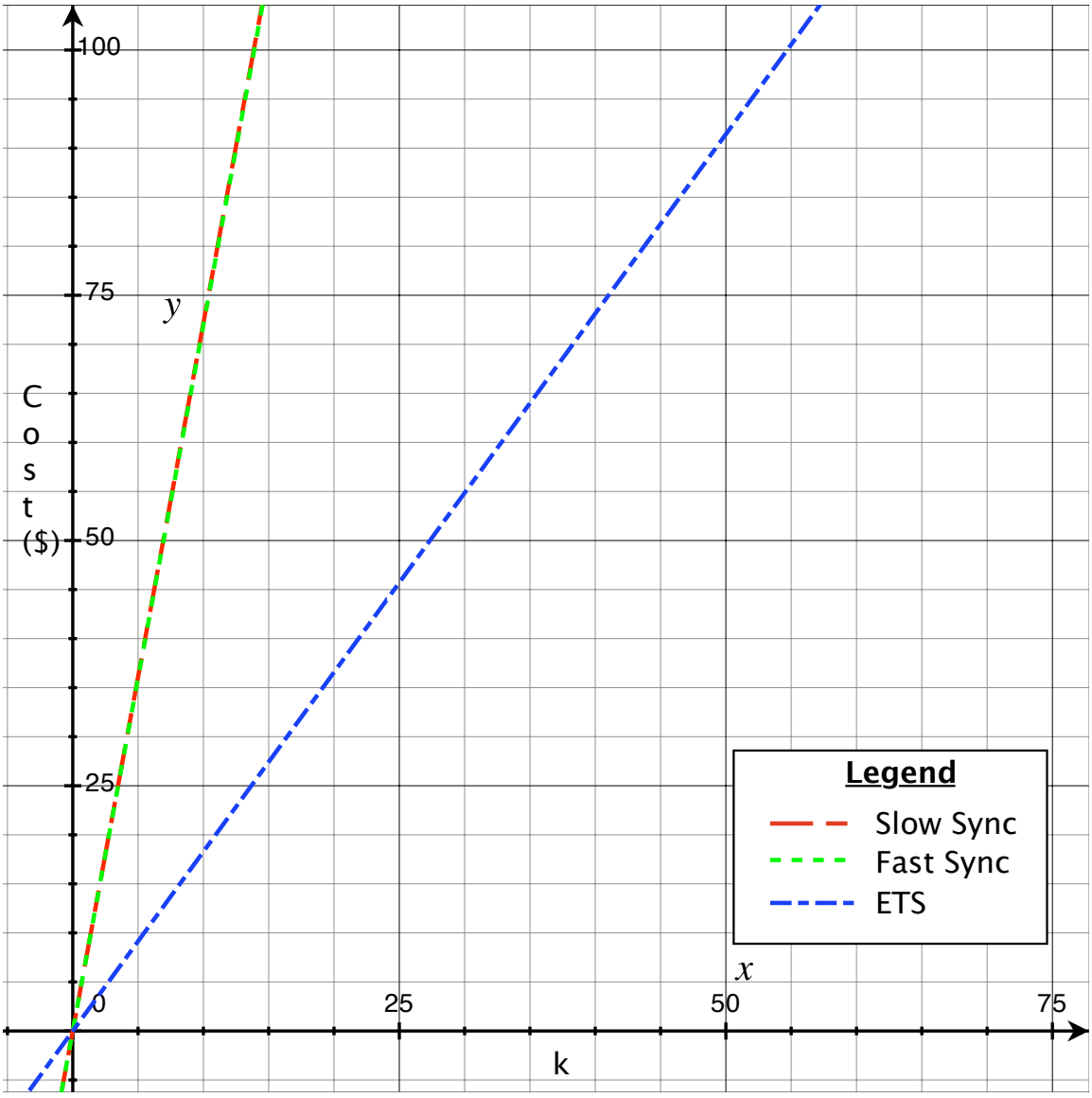

Figure 7.1: Graph displaying the results for Simple User A, using the Slow Sync, Fast Sync, and ETS algorithms. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

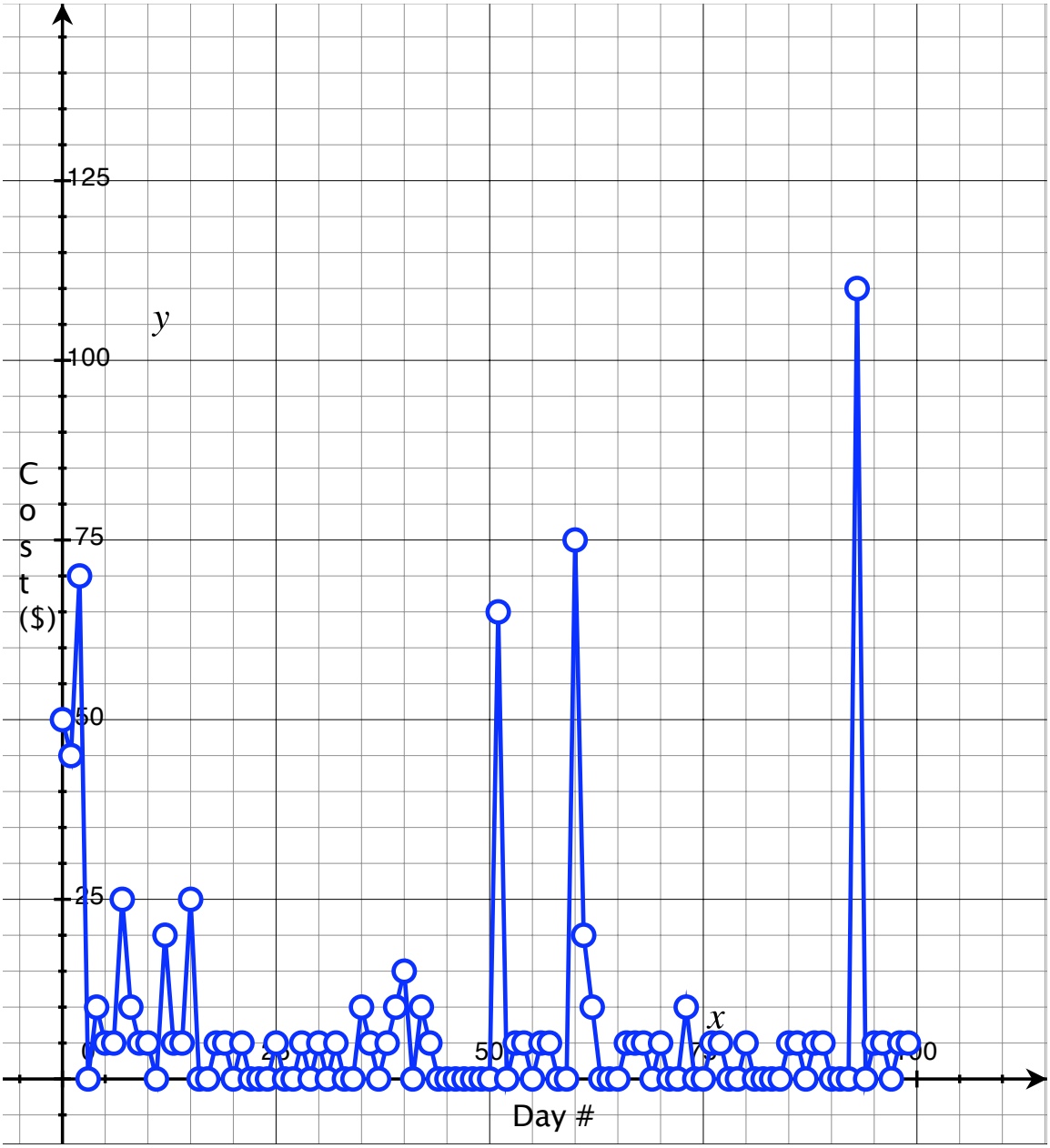

Figure 7.2: Graph displaying the daily ETS costs for Simple User A for a 100-day run. In this graph, the x axis represents the simulated day number, and the y axis represents the cost of synchronization.

When we attempt to evaluate the ETS algorithm against the Simple User B user model against varying estimated average out-of-date record access cost factors k, where $k \in \{0.1, 1, 10, 100, 1000, 10000\}$, we can see from figure 7.4 that the results appear to be linear. However, looking at the graph in more detail (figure 7.5), we can see that in fact that the results are not linear to the increases to k. Indeed, looking at the raw results in table G.8, we can see that as k increases, the slope m of the solution line decreases, while the network communications cost b and the average total data transferred both increase. With these results, we can clearly see for this user model the ETS algorithm making the trade-off between minimizing out-of-date record accesses and the network cost of synchronization.

A sample daily cost for a 100 day run of the ETS algorithm for Simple User B can be seen in figure 7.6. Once again, we can see that the ETS algorithm quickly converges upon a relatively tight solution range. It is interesting to note, however, that the very first value tested appears to have a lower overall cost associated with it than the final convergence range; this can be explained, however, if the initial tested value was sufficiently high that no synchronization occurred, as the ETS algorithm is designed to short-circuit around choices which prevent any synchronization during the day. Being the first day, all records are initially considered to be in full synchronization, and thus only a small number of differences may be both generated and accessed, causing it to display an other low overall cost.

7.3.3 Business User A

Business User A shows good results in favour of the ETS algorithm, with the ETS algorithms cost equation resulting in a lower result slope that either the Slow or Fast Sync algorithms, signifying fewer average out of date record accesses. See figure 7.7. As expected, both the Fast Sync and Slow Sync algorithms have approximately the same slope, due to sharing the same average synchronization rate, however the Fast Sync algorithm does exhibit a lower network cost. This is also to be expected,

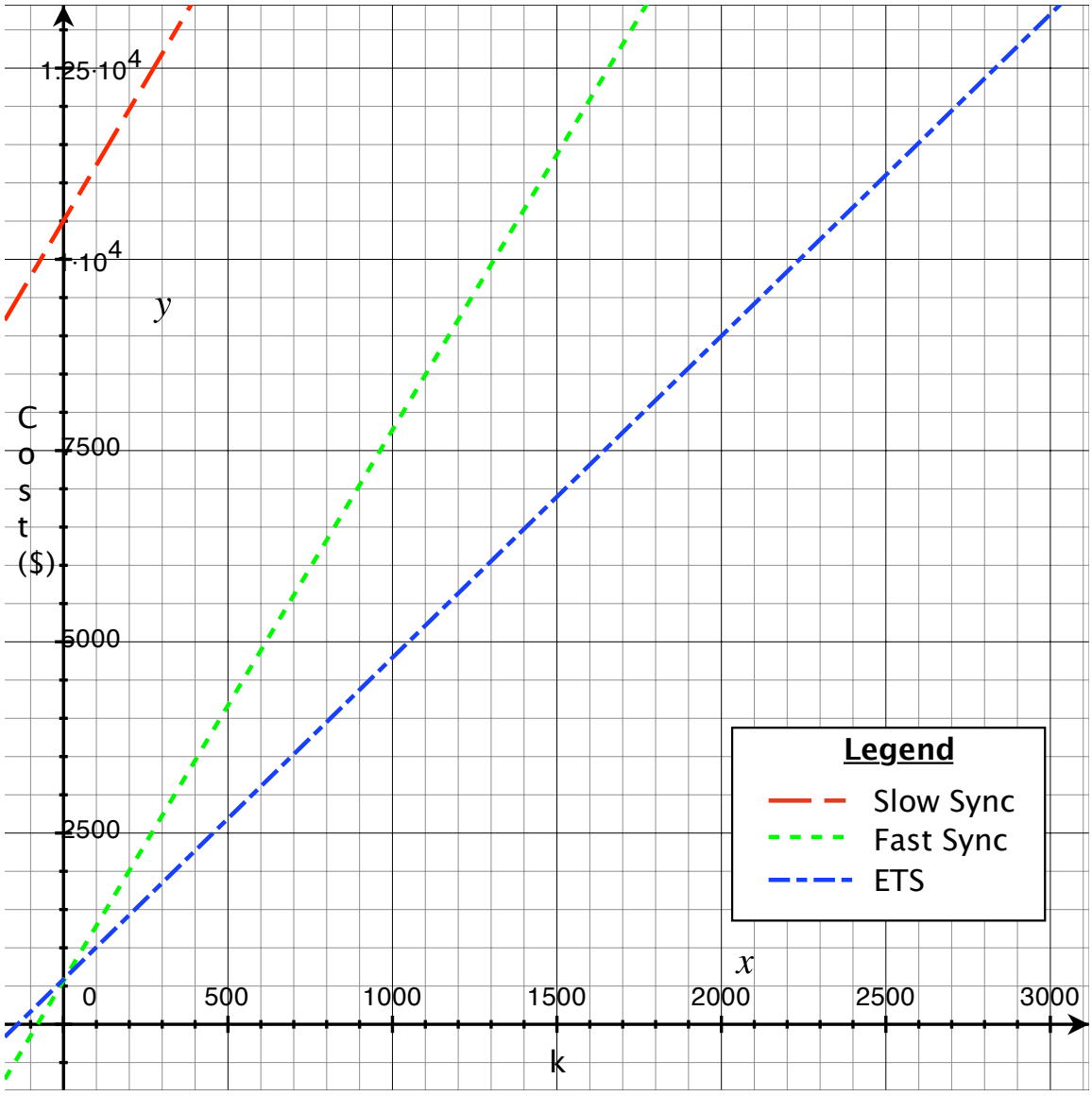

Figure 7.3: Graph displaying the results for Simple User B, using the Slow Sync, Fast Sync, and ETS algorithms. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

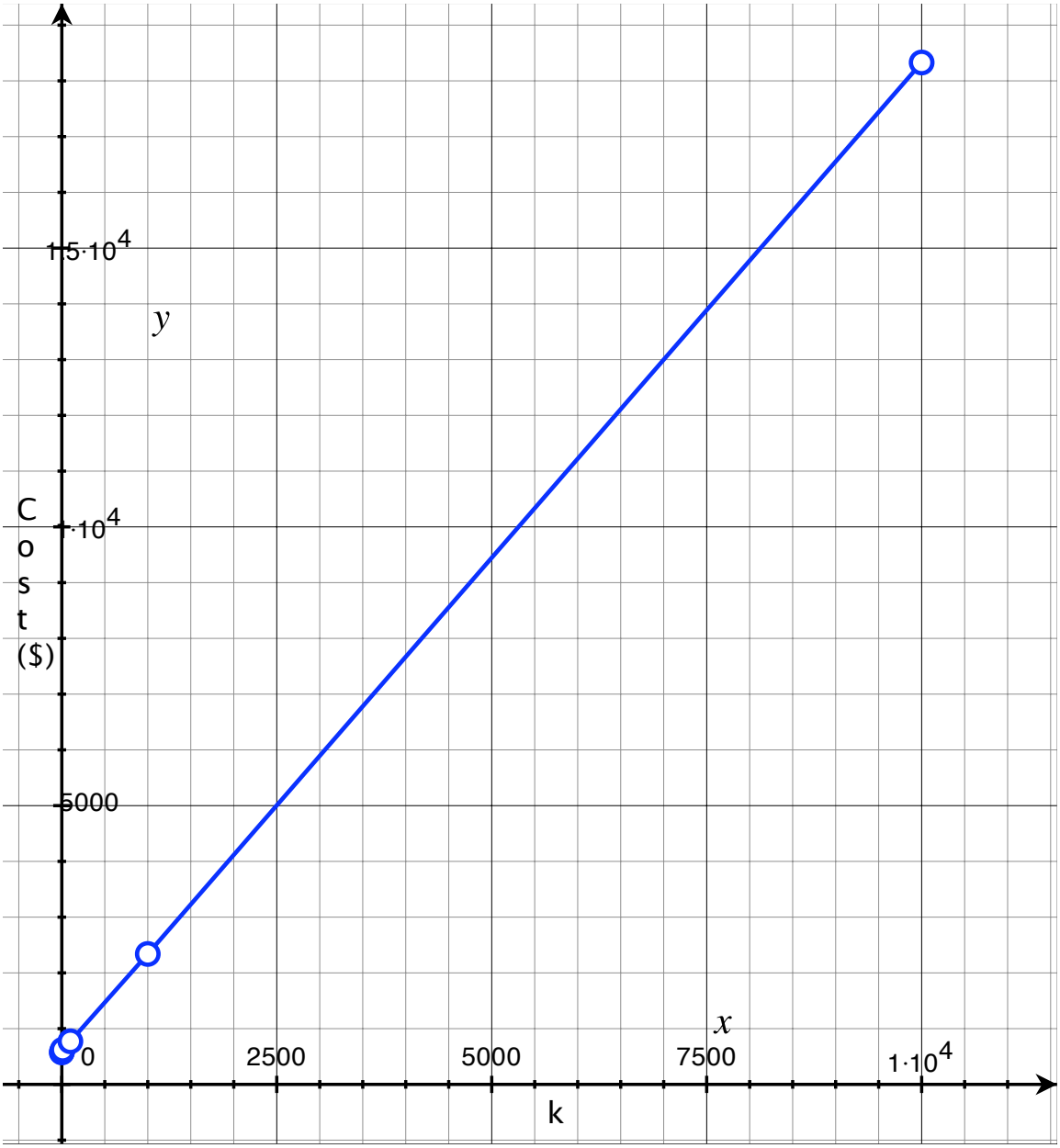

Figure 7.4: Graph displaying ETS results for Simple User B, for $k \in \{0.1, 1, 10, 100, 1000, 10000\}$. In this graph, the x-axis represents k, while the y-axis represents the average cost of synchronization using the ETS algorithm.

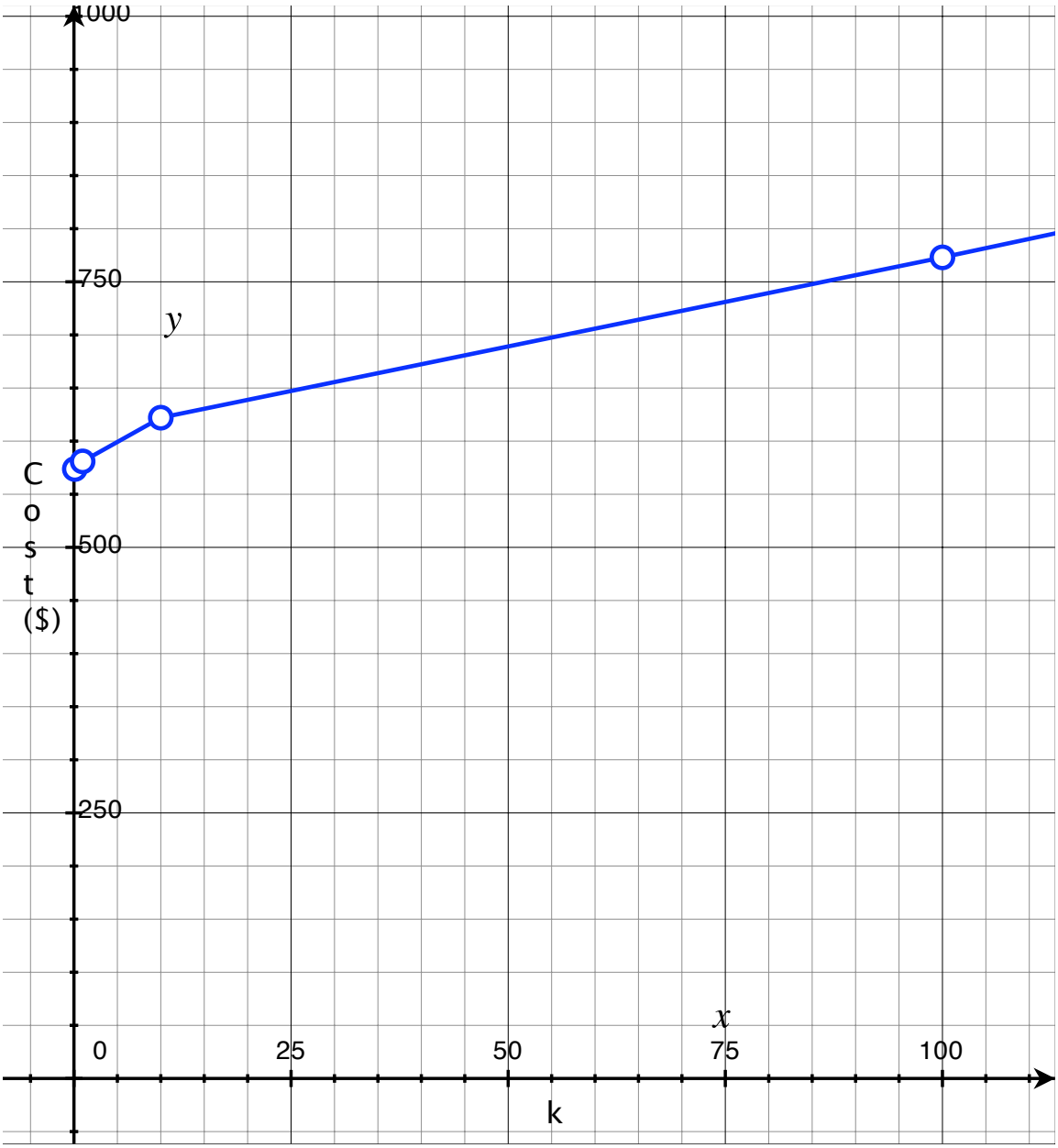

Figure 7.5: Graph displaying ETS results for Simple User B, for $k \in \{0.1, 1, 10, 100\}$. In this graph, the x-axis represents k, while the y-axis represents the average cost of synchronization using the ETS algorithm

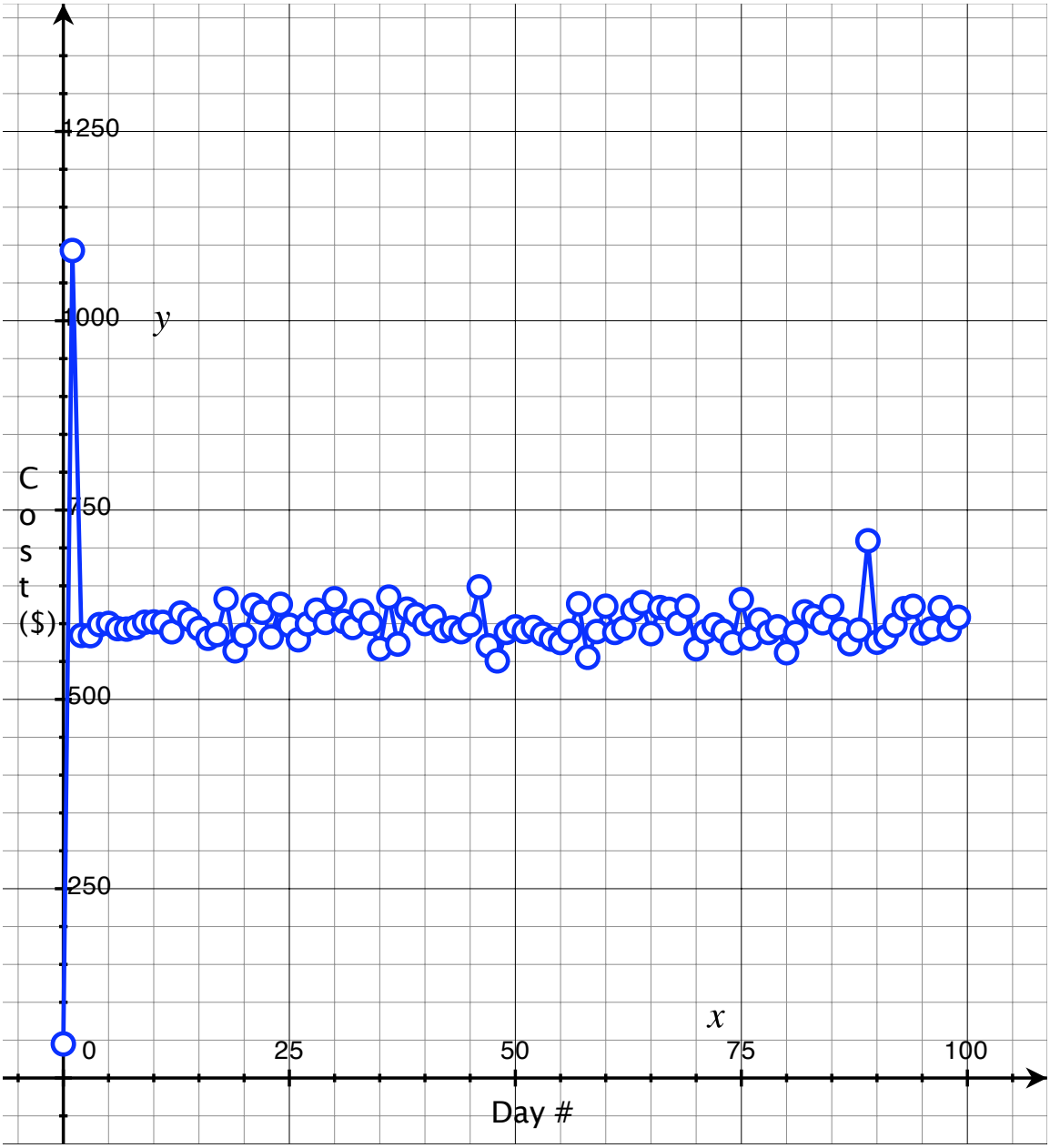

Figure 7.6: Graph displaying the daily ETS costs for Simple User B for a 100-day run. In this graph, the x axis represents the simulated day number, and the y axis represents the cost of synchronization.

as the nature of the Fast Sync algorithm is to synchronize less data during each synchronization session. While the ETS algorithm does exhibit a higher average daily network access cost that either the Slow Sync or Fast Sync algorithms, at the $5 estimated out-of-date record access cost used for evaluating the ETS algorithm, the ETS algorithm costs less in actual dollar value. Using this $5 estimate, the Slow Sync algorithm would cost this user an average of $36.06 per day, the Fast Sync algorithm would cost this user an average of $30.838 per day, whereas the ETS algorithm would cost this user an average of $29.736 per day. See table G.3 for the full results for Business User A.

A sample daily cost for a 100 day run of the ETS algorithm for Business User A can be seen in figure 7.8. Much like Simple User A, Business User A very quickly settles into a low cost range, however there are a variety of random spikes, which negatively affect the average daily cost.

7.3.4 Business User B

In contrast to Business User A, Business User B, who synchronizes once per hour when using the Fast Sync and Slow Sync algorithms, sees the ETS algorithm fare worse when compared to the performance of the other two algorithms in terms of the average number of out-of-date record accesses per day. Where the ETS algorithm shows improvement is in terms of average daily network cost, where it bests the Fast Sync algorithm by a factor of approximately 2.428. Considering that the average cost for missed information input into the ETS algorithm for this user model was $5, the ETS algorithm at this average cost factor was roughly 1.453 times less for ETS versus Fast Sync ($29.74 versus $43.22 respectively), and 170.161 times less for ETS versus Slow Sync (which cost an average of $5059.92 per day).

Figure 7.9 displays the results for this experiment. See table G.4 for the full results for Business User B.

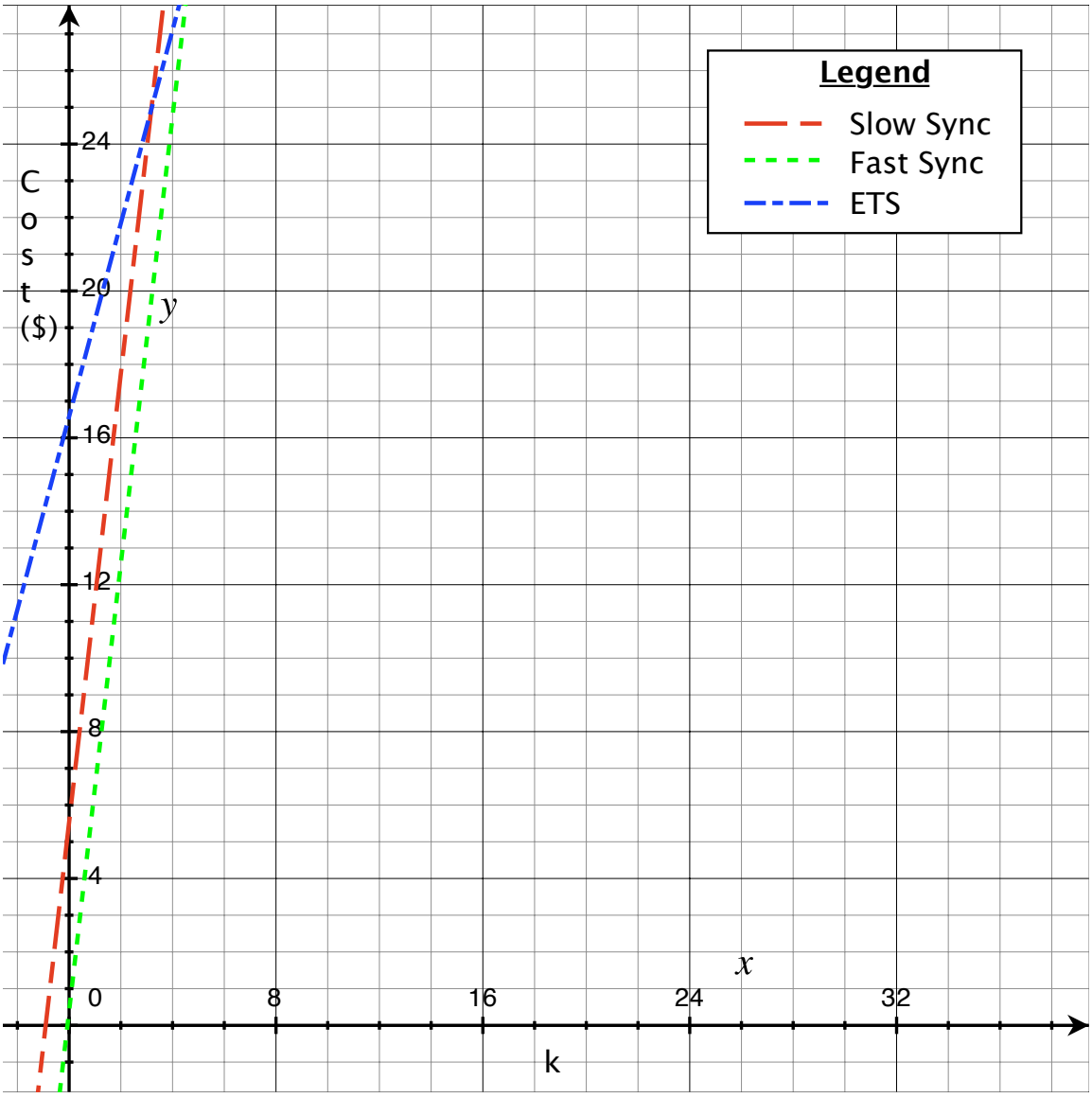

Figure 7.7: Graph displaying the results for Business User A, using the Slow Sync, Fast Sync, and ETS algorithms. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

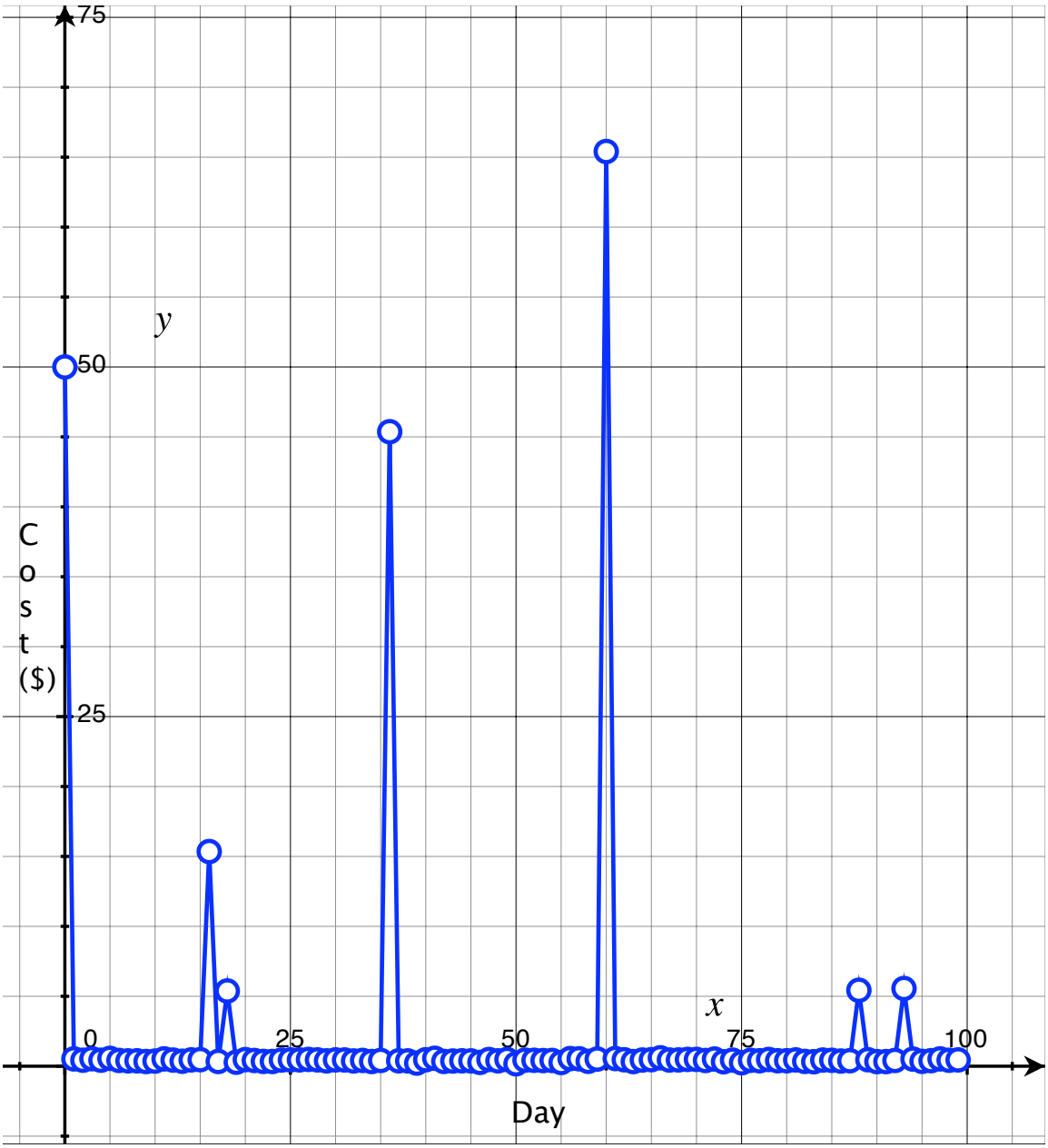

Figure 7.8: Graph displaying the daily ETS costs for Business User A for a 100-day run. In this graph, the x axis represents the simulated day number, and the y axis represents the cost of synchronization.

A sample daily cost for a 100 day run of the ETS algorithm for Business User B can be seen in figure 7.10. As Business User B has the same daily routine as Business User A, to provide better contrast for the 100 day run with Business User A we have set the expected average out-of-date record cost k to \$50, a ten-fold increase over Business User A. The results are nearly identical to Business User A, however what spikes do appear to occur within the costs appears to be more muted. While the decreased magnitude of these spikes may be due to the increased value of k, it may also simply be due to a lack of any random convergences in the 100 day run that have caused significant spikes in the other models. As such, this lack of significant spikes may have less to do with k, than with good fortune while the test was being executed.

7.3.5 Business User C

Contrasting again against Business User A, Business User C demonstrates some of the issues the synchronization algorithms display as record size increases, while all other variables remain the same. As can be seen in G.5, the results for the Null Sync algorithm are virtually identical to those seen in the results for Business User A and Business User B; lacking any synchronization, and identical variables for access probabilities and server-side updates, large records go stale at the same rate as smaller records, and the final costs are statistically equivalent.

Stale records appear to become a significant issue for the Slow Sync algorithm in this experiment. Whereas in the cases of Business User A and B the Slow Sync and Fast Sync algorithms exhibit a roughly equivalent slope, in this experiment the slope of the Slow Sync algorithm results is approximately 2.6 times steeper. The reason for this is evident in the average amount of data exchanged per day for each algorithm: where the Fast Sync algorithm transmits approximately 3.7 GB of information daily, the Slow Sync algorithm transmits an approximate average of 48.4 GB of data each day, roughly 13 time more data. The time to transmit this much

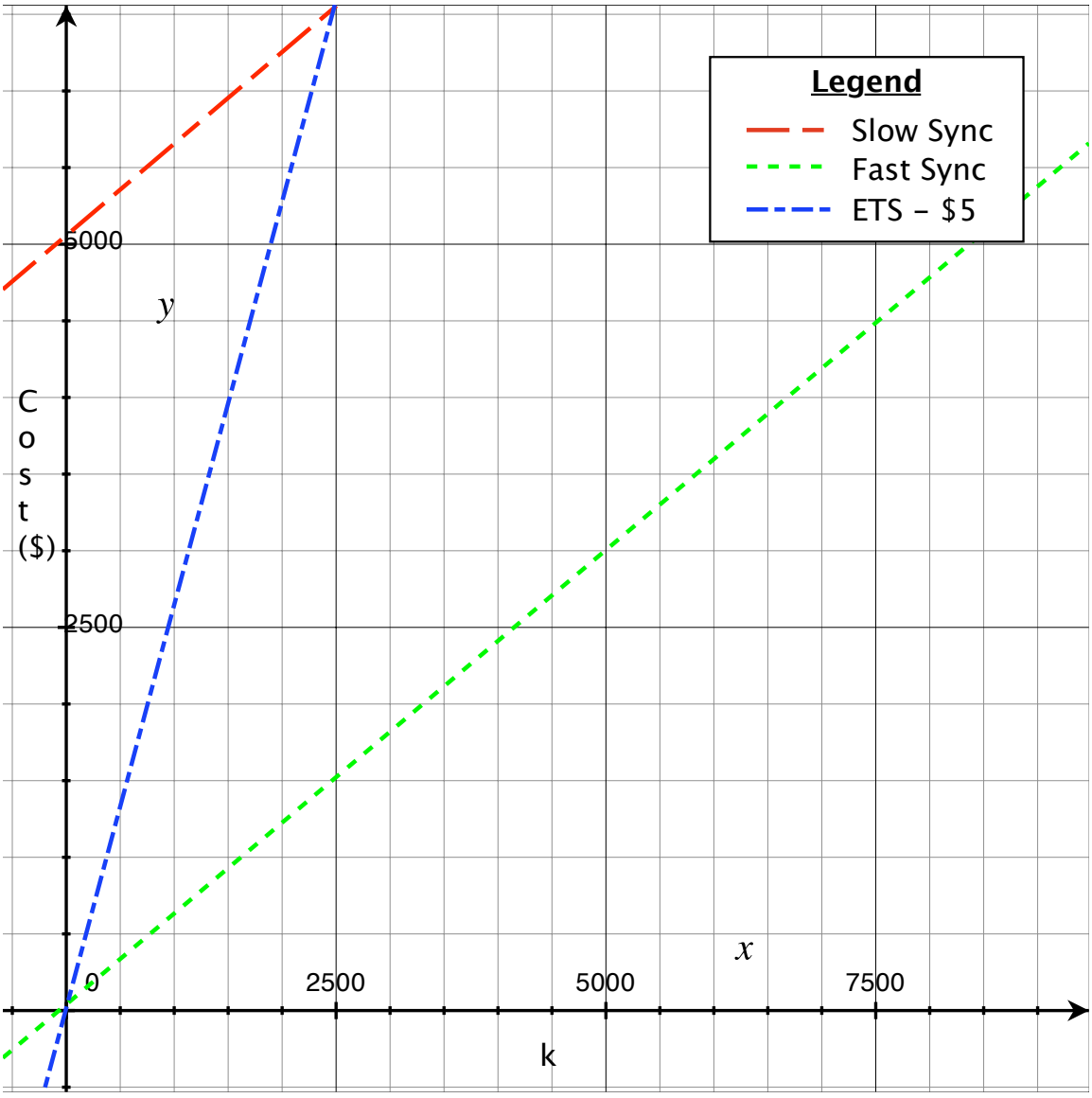

Figure 7.9: Graph displaying the results for Business User B, using the Slow Sync, Fast Sync, and ETS algorithms. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

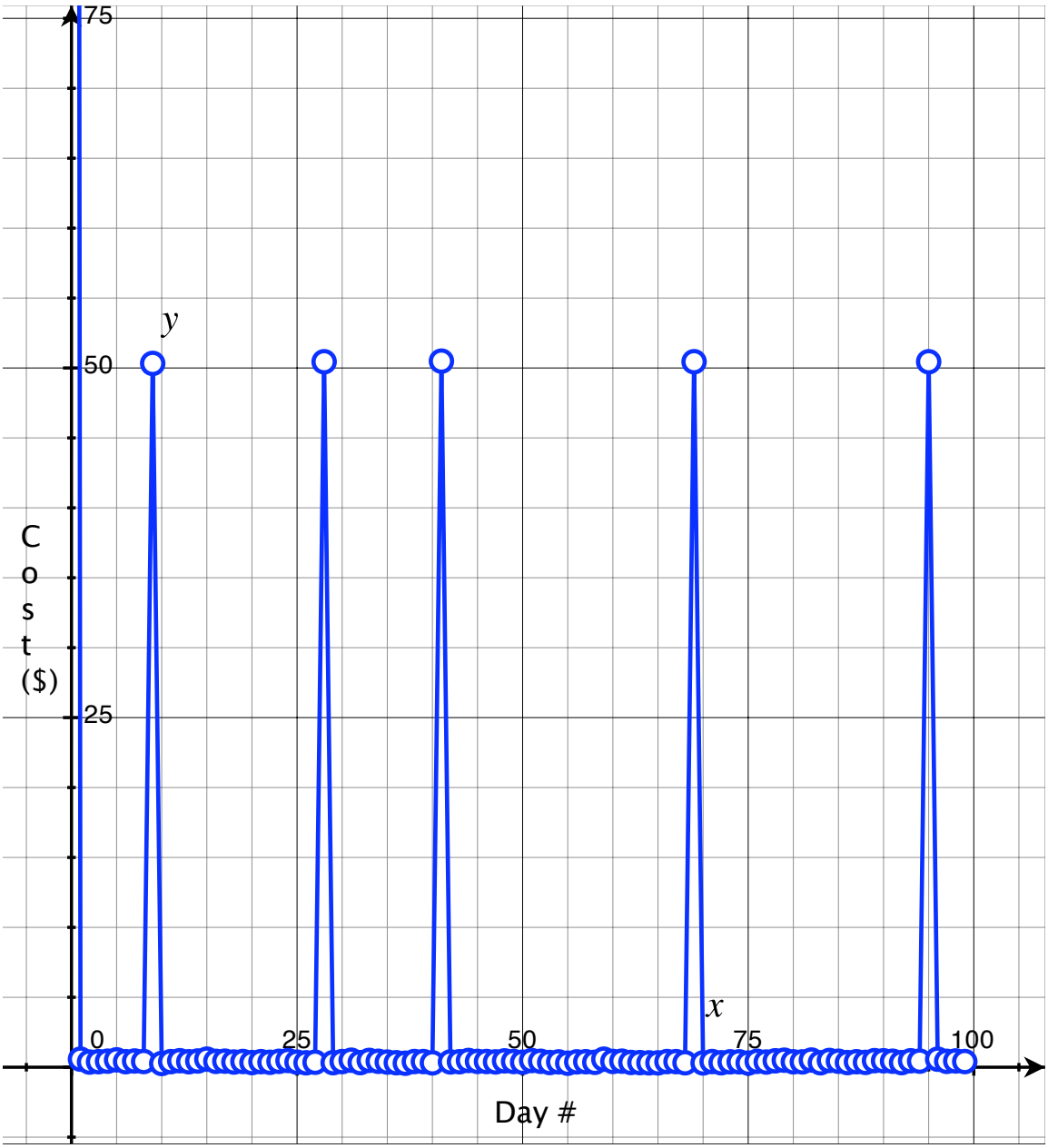

Figure 7.10: Graph displaying the daily ETS costs for Business User B for a 100-day run. In this graph, the x axis represents the simulated day number, and the y axis represents the cost of synchronization.

information is significant, and thus it would appear that the Slow Sync algorithm is missing opportunities it would otherwise take to perform an synchronization, due to it not having completed the previous synchronizations session.

The ETS algorithm shows mixed results in this test scenario. On average, it exchanges roughly 8% more information per day that the Fast Sync algorithm, and yet on average keeps data 4.6 times as fresh. However, this improvement over the Fast Sync algorithm comes at the price of a nearly 35 times higher average daily network access cost.

Figure 7.11 displays the results for this experiment. See table G.5 for the full results for Business User B.

7.3.6 Mobile Salesperson A

Mobile Salesperson A synchronizes on average four times per hour while the device is powered on, which is the maximum per-hour rate that the ETS algorithm was designed to support (as discussed in section 5.2). Because of this high rate of synchronization, it is expected that both the Fast Sync and Slow Sync algorithms will have a relatively low incidence of out-of-date record accesses, at the expense of high network access cost. This is reflected in the experimental results as presented in figure 7.12, where we can see that both the Slow Sync and Fast Sync algorithms have a low slope, indicating the average out-of-date record accesses, while simultaneously featuring a relatively high y-axis intersection, indicating a high average daily network access cost. The ETS algorithm, in contrast, sacrifices being highly up-to-date in exchange for a savings in network access cost. At the ETS target out-of-date access cost of $10, this translates to an average per-day cost of $14134.63 for the Slow Sync algorithm, $97.17 for the Fast Sync algorithm, and $85.21 for the ETS algorithm. At the estimated $10 out-of-date access cost, the solution automatically derived by the ETS algorithm continues to be competitive to the Fast Sync algorithm up to an

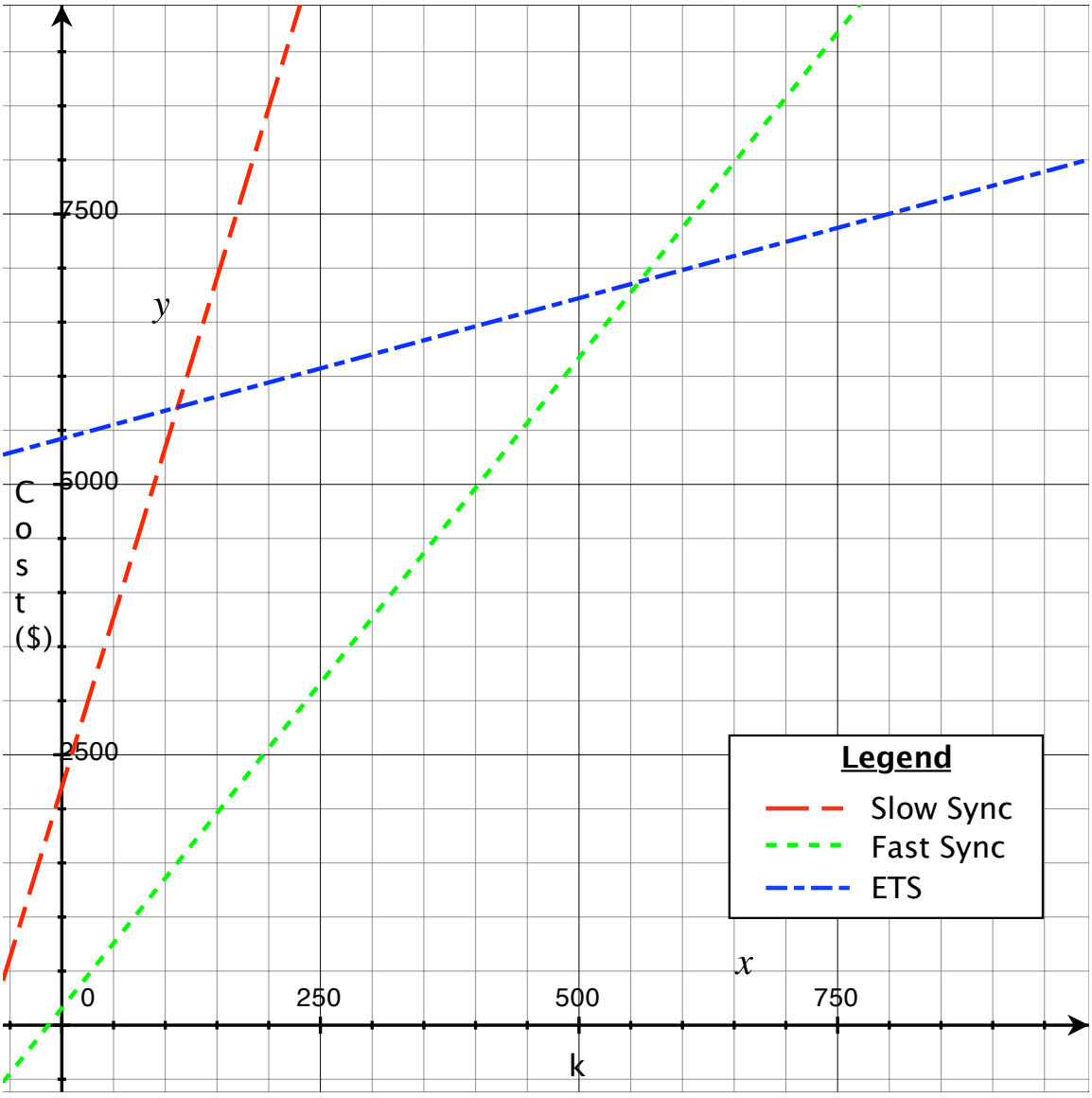

Figure 7.11: Graph displaying the results for Business User C, using the Slow Sync, Fast Sync, and ETS algorithms. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

out-of-date access cost of $11.81, and is competitive with the Slow Sync algorithm up to an out-of-date access cost of $2133.87.

See table G.6 for the full results for Mobile Salesperson A.

When we attempt to evaluate the ETS algorithm against the Mobile Salesperson A user model against varying estimated average out-of-date record access cost factors k, where $k \in \{0.1, 1, 10, 100, 1000, 10000\}$, we can see from figure 7.13 that all six results are approximately linear; looking at the raw results in table G.9 we can see that while the slopes for all six solutions are virtually identical, there is some indication that as k increases, the overall network access cost is likewise increasing. This would appear to indicate an attempt on the part of the ETS algorithm to trade off an increasing frequency of synchronization in the more expensive networks to decrease out-of-date record accesses, however for unknown reasons this appears to be failing. It would appear that the algorithm is frequently falling into a local minima, from which improvements are difficult to come by.

A sample daily cost for a 100 day run of the ETS algorithm for Mobile Salesperson A can be seen in figure 7.14. Like most of the previous models, the ETS algorithm when run against Mobile Salesperson A appears to generally converge, however once again we see a number of random outliers which significantly negatively impact the overall average cost of synchronization.

7.3.7 Truck Driver A

Truck Driver A appears to present a problem for the ETS algorithm: while the ETS algorithm initially outperforms both the Slow Sync and Fast Sync algorithms, its higher slope (representing the average number of out-of-date record accesses) is higher than both, leading it to eventually out-cost both algorithms when used on an hourly basis. This is especially problematic as for the ETS algorithm, the cost of out-of-date information for this user model was estimated to be $250, however the ETS algorithm's use cost exceeds that of the Fast Sync algorithm when the cost of

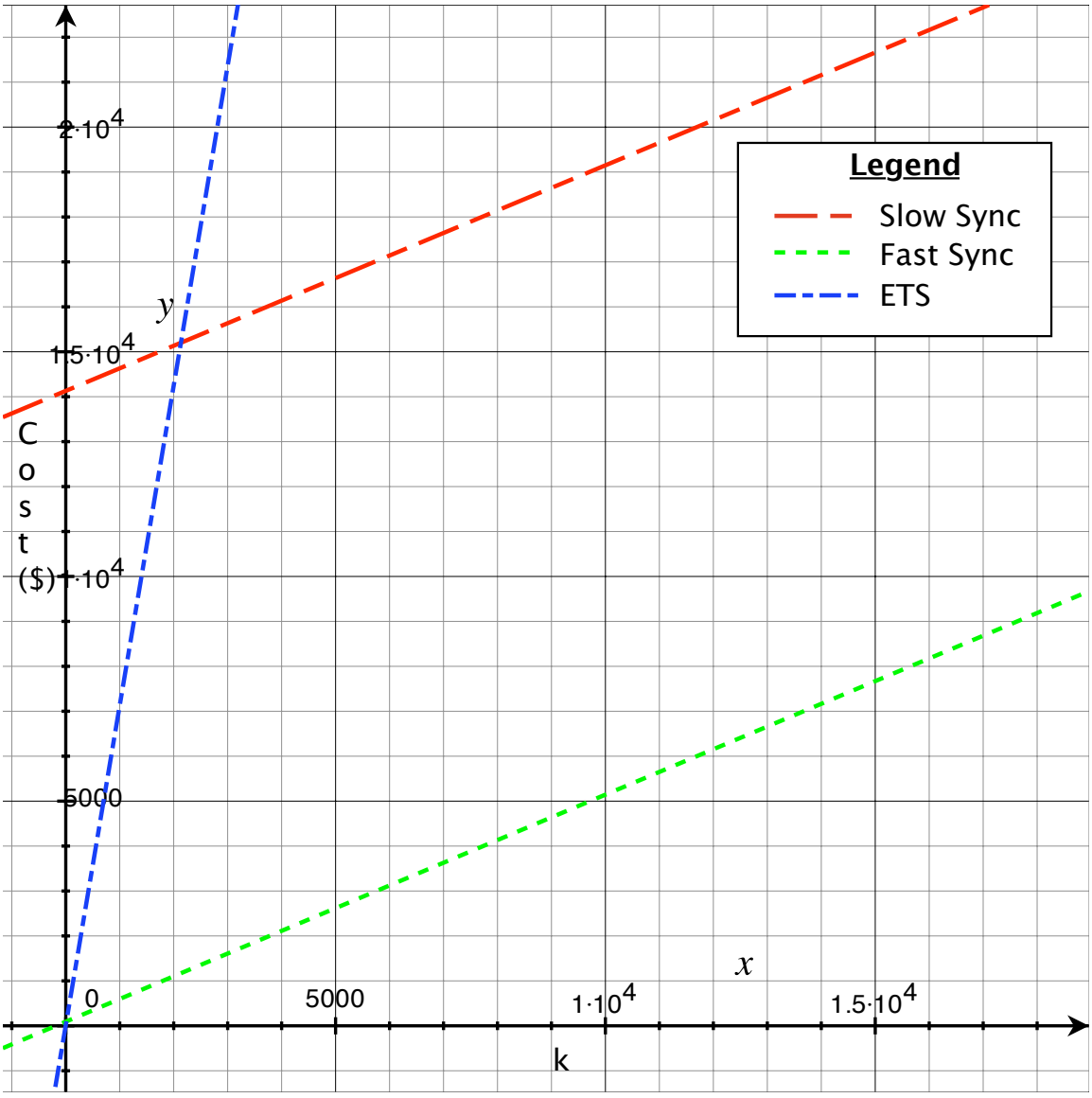

Figure 7.12: Graph displaying the results for Mobile Salesperson A, using the Slow Sync, Fast Sync, and ETS algorithms. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

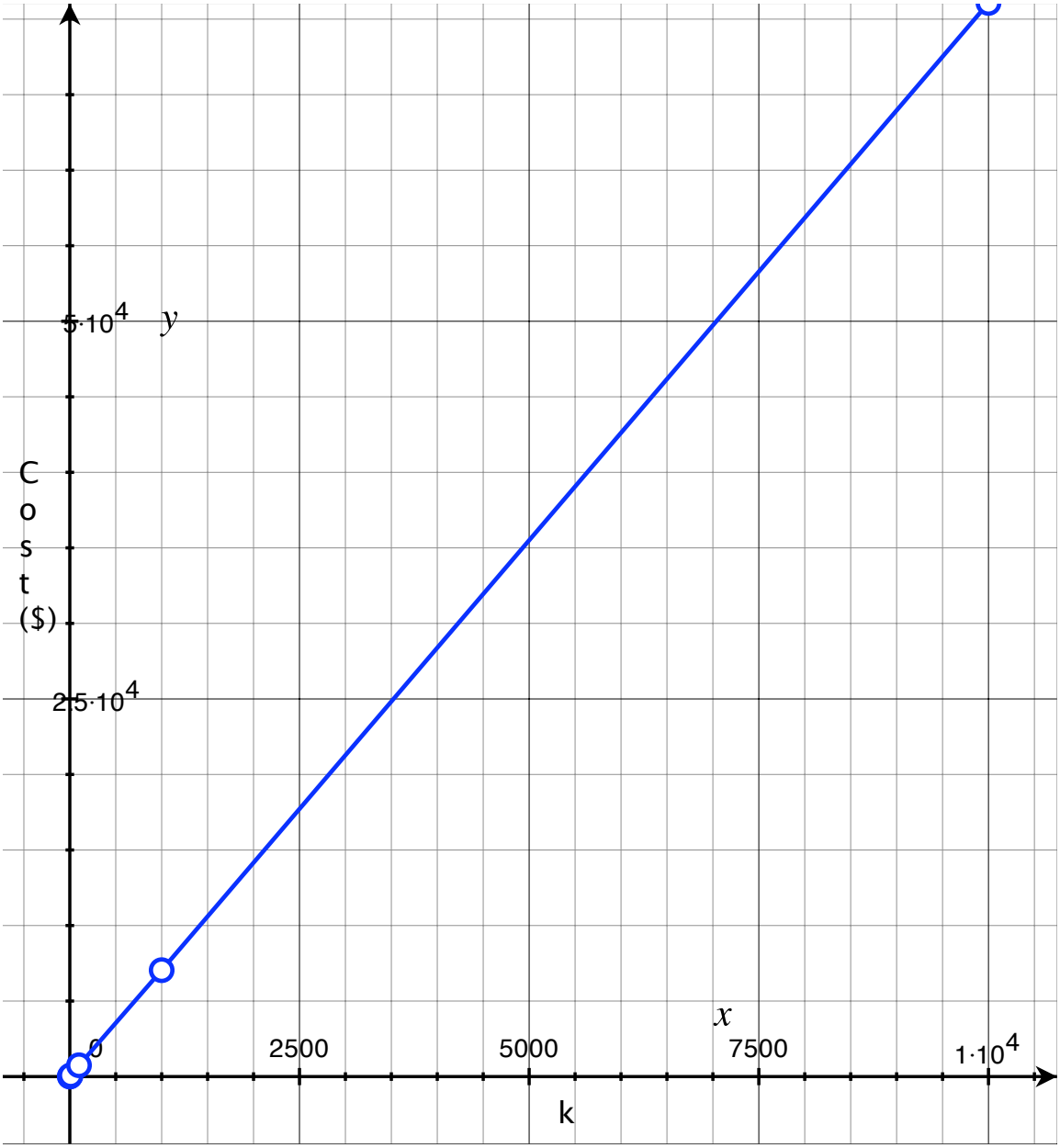

Figure 7.13: Graph displaying ETS results for Mobile Salesperson A, for $k \in \{0.1, 1, 10, 100, 1000, 10000\}$. In this graph, the x-axis represents k, while the y-axis represents the average cost of synchronization using the ETS algorithm, evaluated at k

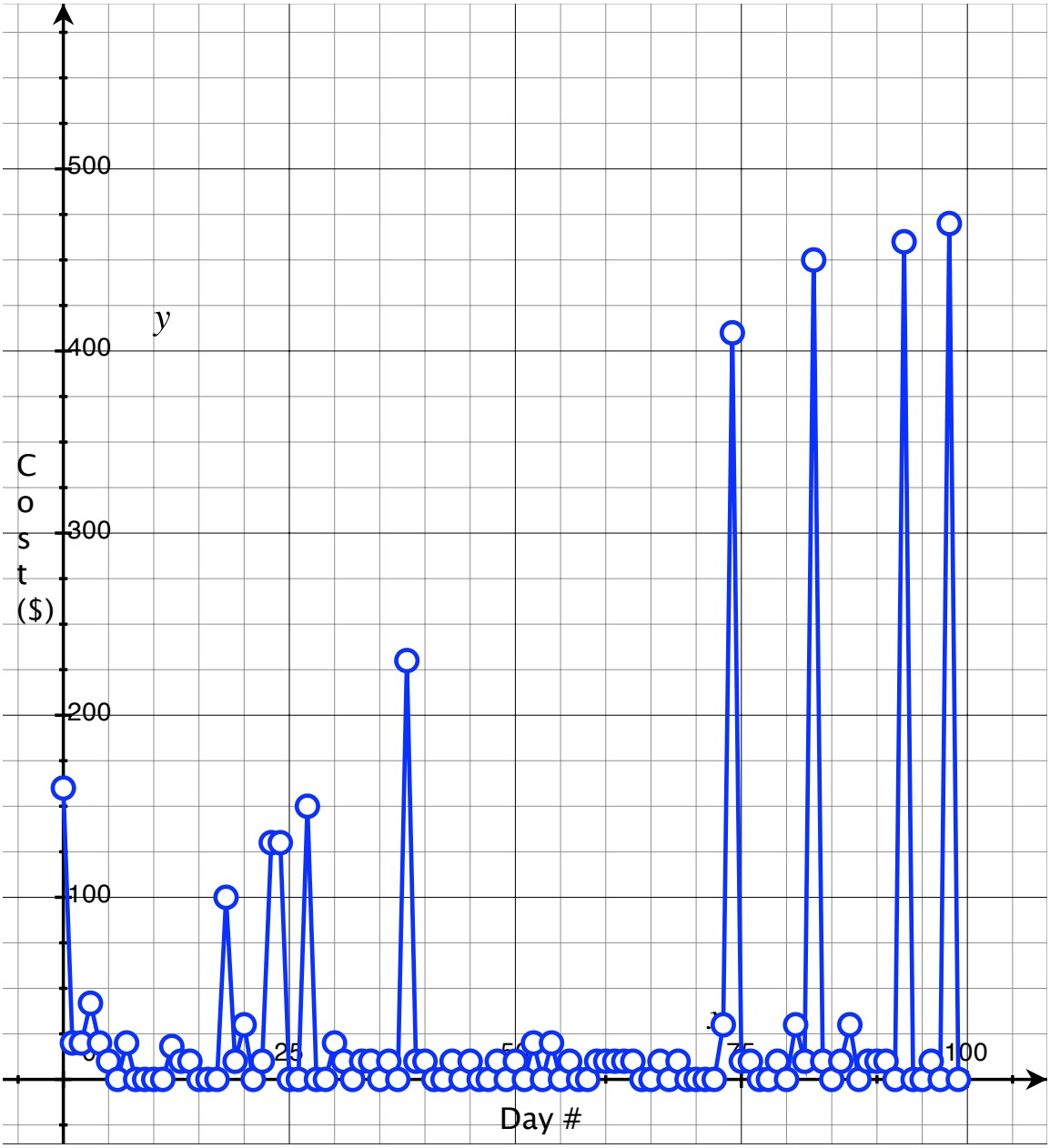

Figure 7.14: Graph displaying the daily ETS costs for Mobile Salesperson A for a 100-day run. In this graph, the x axis represents the simulated day number, and the y axis represents the cost of synchronization.

using out-of-date information is greater than approximately $20.86. At the target average out-of-date record access cost of $250, Slow Sync would cost this user, on average, $7262.82 per day, Fast Sync would cost an average of $250.05 per day, and the ETS algorithm would cost an average of $620.57 per day. Figure 7.15 displays the results for this experiment. See table G.7 for the full results for Truck Driver A.

A sample daily cost for a 100 day run of the ETS algorithm for Truck Driver A can be seen in figure 7.16. These results once again appear to mirror those of the previous algorithms, with general convergence with a few random spikes which increase the average daily cost of synchronization with the ETS algorithm.

7.4 Analysis of Results

In all six experiments, the ETS algorithm easily beat the Null Sync and Random Sync artificial control algorithms, as well as the Slow Sync algorithm, which suffers from too much network use, driving up its use cost significantly. As demonstrated in three of the six experiments, the ETS algorithm can provide noticeable improvements in average daily synchronization cost over the Fast Sync algorithm (as in the cases of Simple User A, Business User B, and Sales Person A), moderate improvements over the Fast Sync algorithm in one of the experiments (Business User A), nearly identical results to the Fast Sync algorithm in one of the experiments (Simple User B), and moderately worse results than the Fast Sync algorithm in one of the experiments (Truck Driver A). In all of these experiments, however, the ETS algorithm was able to beat the Fast Sync algorithm in either the average number of out-of-date record accesses per day (as is the case with Simple Users A and B, and Business User A), or it beat the Fast Sync algorithm in terms of average daily network access cost (as is the case with Business User B, Sales Person A, and Truck Driver A).

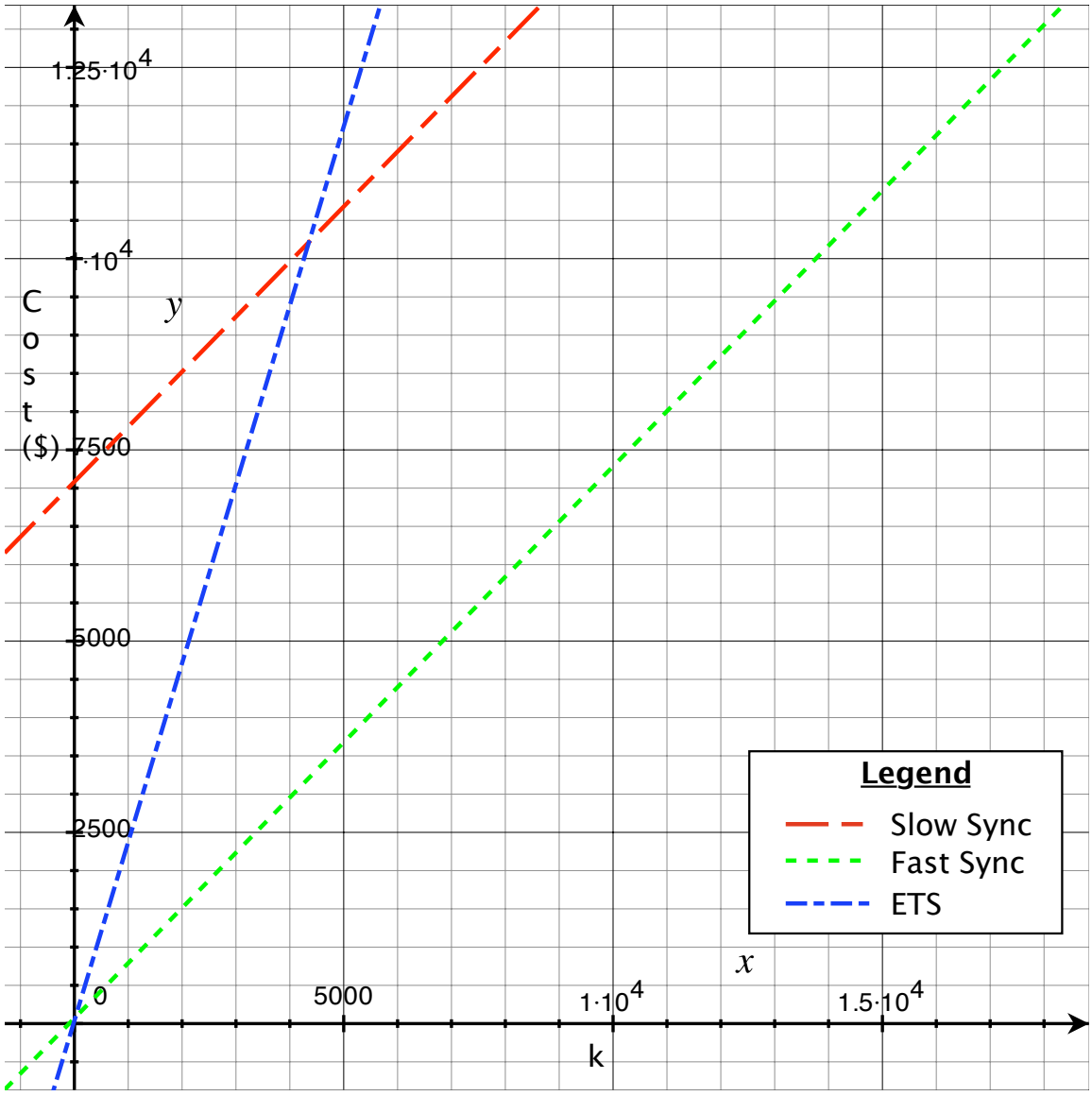

Figure 7.15: Graph displaying the results for Truck Driver A, using the Slow Sync, Fast Sync, and ETS algorithms. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

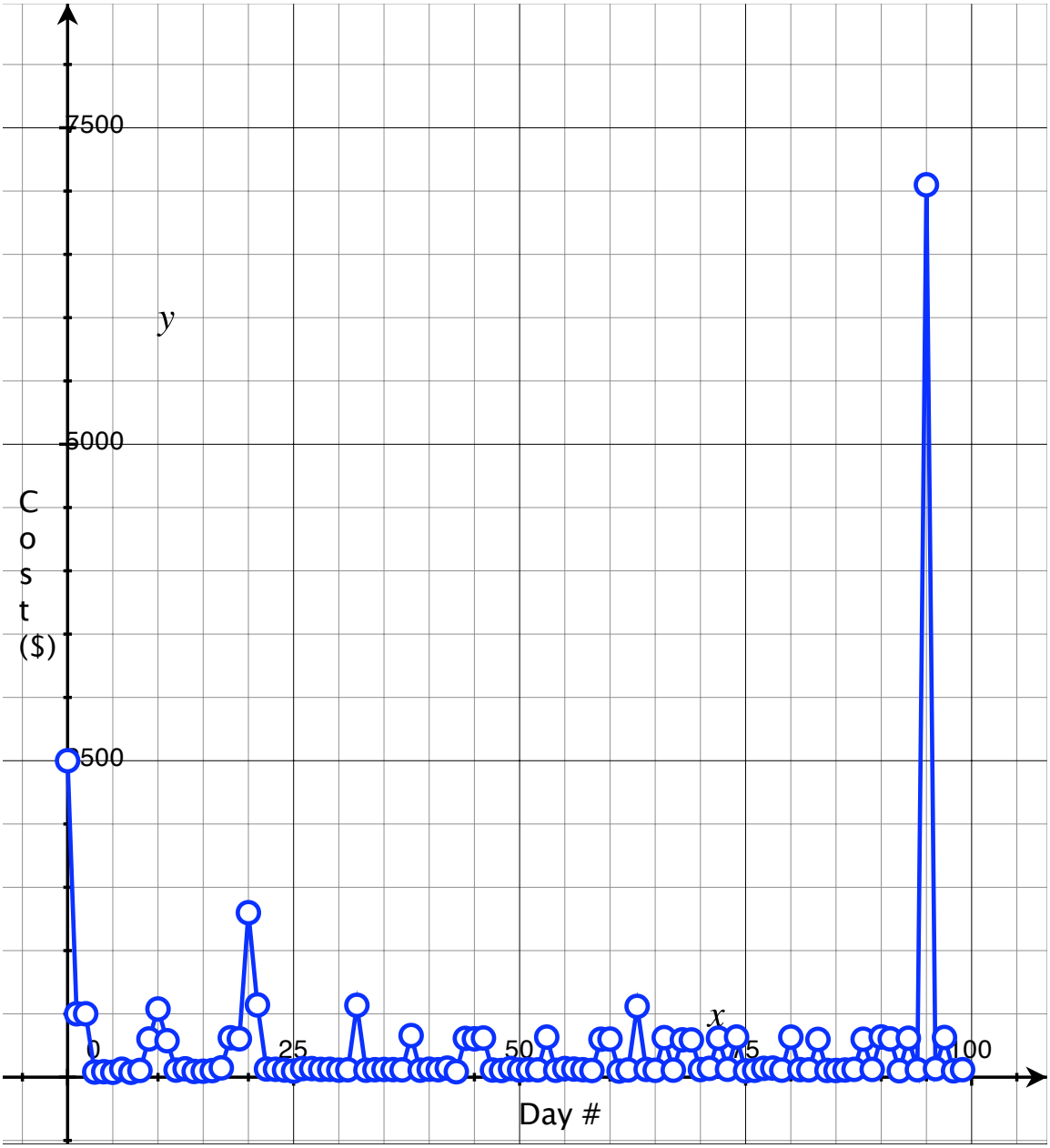

Figure 7.16: Graph displaying the daily ETS costs for Truck Driver A for a 100-day run. In this graph, the x axis represents the simulated day number, and the y axis represents the cost of synchronization.

As is to be expected, as evidenced by figures 7.2, 7.6, 7.8, 7.10, 7.14, and 7.16, the ETS algorithm frequently starts off with a high cost profile, but eventually converges towards a lower cost profile.

Figure 7.17 shows two graphs, comparing Simple User B's cost averages for the Fast Sync algorithm and ETS algorithms, with darker shaded areas signifying the daily cost ranges out to one standard deviation. It should be noted from this graph that while the ETS algorithm has a greater range of variation, if we calculate the specific standard deviation values for the estimated average out-of-date cost value for this model ($5), the Fast Sync algorithm has a standard deviation of ±190.920, whereas the ETS algorithm has a standard deviation of ±93.937.

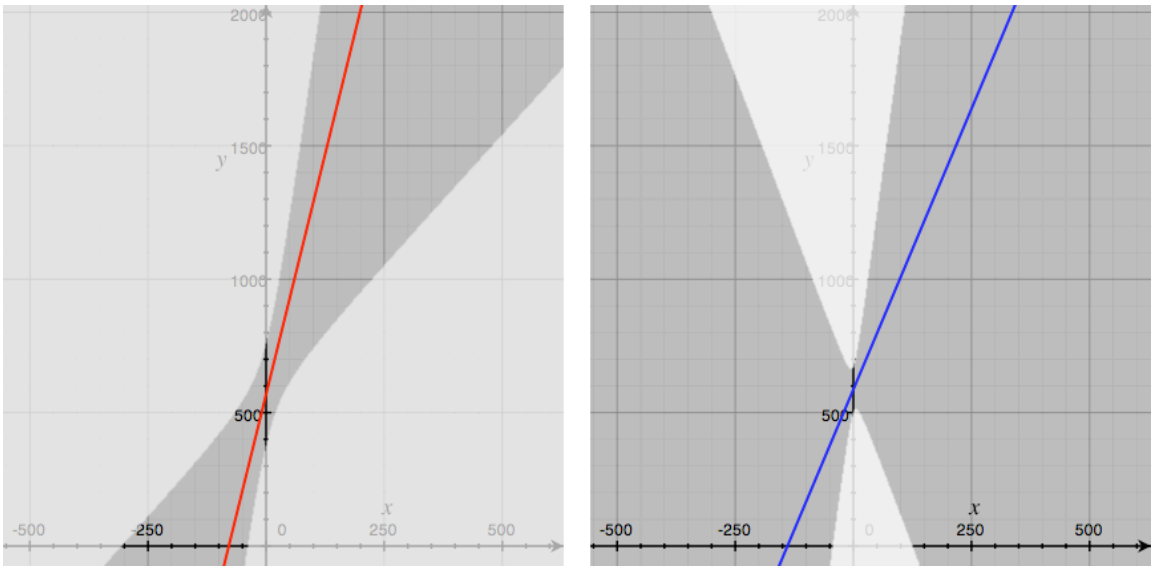

Figure 7.17: Graphs comparing Fast Sync (left) to ETS (right) at 1 Standard Deviation, based on the results for Simple User B. In this view, the darker shaded area constitutes the range up to one standard deviation from the average cost line. In this graph, the x axis represents possible values for k, and the y axis represents the average daily synchronization cost.

A special area of concern is that of convincing the user to trust the ETS algorithms ability to minimize costs. At the outset, costs may be extremely high as the algorithm evaluates threshold candidates which result in either huge network costs, or significant

out-of-date information. Because of this, it would be easy for some users to conclude after a week of use that the algorithm is very expensive and untrustworthy to use.

Another area of concern is demonstrated by comparing the ETS results for Business User A and Business User C. In these two cases, the latter experiment uses a mean record size nearly 410 times larger than the former, however both exhibit nearly equivalent slopes, indicating a near-identical effort to keep data fresh. Where the ETS algorithm fails in relation to Business User C is in terms of the network cost, which is significantly higher than that of the Fast Sync algorithm. Note, however, that for Business User C the average network access cost is only approximately 327 times that of Business User A, which is about 80% of the average record size difference, indicating that the ETS algorithm has achieved some improvements in this regard. However, Business User C indicates that it is possible that record size, and consequently the cost to synchronize an individual record may not weigh heavily enough in the ETS algorithm.

Overall, the results are felt to be positive in favour of the ETS algorithm as a cost-reducing mechanism for mobile device data synchronization, however it it not without its flaws. As the example of Truck Driver A demonstrates, the algorithm may have a difficult time finding a minimum value that is better than the existing algorithms. Additionally, the presence of random spikes in the 100-day graphs (figures 7.2, 7.6, 7.8, 7.10, 7.14, and 7.16) is cause for concern, as poor threshold update decisions may be made based upon these random outliers. Possible areas of further research to improve upon this thesis are details in section 8.1 below.

Chapter 8

Conclusions

As evidenced by the results, the ETS algorithm shows promise in reducing the cost of data synchronization over the existing Fast Sync and Slow Sync protocols, however it does demonstrate some problem areas which we identify as possible areas of future research. To conclude this thesis, ideas for future research are discussed, along with my final conclusions.

8.1 Future Research

There are a variety of areas in which the ETS algorithm can be improved, to provide additional and more consistent cost savings across a variety of different user profile types. There are also a number of existing limitations with the simulation environment, and as such the application of the ETS algorithm's results in this thesis cannot be extrapolated to real-world users.

Firstly, the current simulation environment continually runs the same day pattern repeatedly, with fixed locale entry times that never differ, for the entirety of the simulation run. This rigid pattern does not map particularly well to real-world users, where entry and exit times may be variable, and where they exhibit different patterns on different days. While the ETS algorithm was designed to take some of these issues into account, they were not tested in this thesis due to this limitation in the simulation environment. Further research and development is needed to improve the simulation

environment, to make it capable of handling something closer to real-world user patterns.

Similarly, the current simulation also presumes that day-to-day patterns for the last seven days are generally the same 5.2. However, certain classes of users may exhibit different patterns on different days of the week, but which are comparable for the same day of the week. That is, a given user may not have the same daily pattern, but may have a reasonably comparable pattern on Wednesdays. Experiments using the ETS algorithm, extended to record several weeks worth of daily access, with the access probabilities based on those of the same day of the week is another avenue of further research.

As mentioned in 4.6, the results of the experiments are only demonstrably valid for the simple mathematical models selected. Further experiments using other distributions and arrival models, particularly those with more complexity, would be worthwhile. In addition to experiments using other models and distributions within the simulation environment, expanding and completing the user model study is key to improving the evaluation of not only the ETS algorithm, but of other potential intelligent synchronization algorithms. Additionally, it may be possible to use existing wireless data sets to derive accurate user models, such as the data sets available from Dartmouth University's CRAWDAD project [3, 10, 35].

In terms of improvements to the ETS algorithm itself, there are a variety of areas available for further research. Firstly, the random outliers visible in several of the 100-day result graphs (figures 7.2, 7.6, 7.8, 7.10, 7.14, and 7.16) is of concern; in particular, in the current implementation of the ETS algorithm these random outliers could cause the algorithm to make a poor cost-improvement choice. As the ETS algorithm appears to converge quickly based on the results of the 100-day experiments, three potential solutions to this issue are readily apparent. In the first, it should be possible to smooth out the effects of these outliers somewhat by running

the same threshold for multiple days, and basing the threshold update decision on the average of these results. While this would presumably bring any set of results for a single threshold value closer to the norm whenever an outlier occurs, the outlier could still have a significant impact on cost. A second solution would be to again run the same threshold for multiple days, selecting the median value as the cost for evaluation. A third solution would be to blend aspects of the first two solutions together, such that when evaluating the results of a multi-day threshold test, an equal number of high and low outliers are removed from the result set, with the remaining results averaged together to come up with a single cost value, which can then be evaluated.

Another area worthy of further examination is the initial threshold point selection. During development, it was noted at one point that the threshold initialization algorithm (5.2.1) was only being run during the first simulation run; subsequent runs were not engaging this algorithm. For some of the tests (such as Simple User A), the results were significantly better than the results presented in this thesis, whereas for other tests (such as Simple User B) the results were significantly worse (indeed, the bug was detected when it was noted that the cost of synchronization for this model was nearly as bad as that for the Null Sync algorithm, as no synchronization was occurring). While the results with the threshold initialization algorithm were still good for Simple User A, they didn't approach the low cost of the results without this algorithm. As such, further research can be done on the threshold initialization algorithm.

Another research avenue related to initial threshold point selection would be to extend the threshold calculation to include the calculated thresholds from other devices in the synchronization cloud. In essence, the calculate threshold values could themselves be synchronized through a centralized database, mapping between users with similar habits. This could decrease the time the ETS algorithm spends attempt-

ing to optimize the cost, resulting in a lower average daily cost when using the ETS algorithm.

The arbitrary selection of only one synchronization per 15 minute interval may be unsuitable for many user models. This value was selected so as to minimize the synchronizations impact on device power, and to prevent a scenario where a low threshold, necessary to synchronize when in a higher-cost network zone, could cause constant synchronization when in a virtually free network zone. It was deemed desirable when creating the ETS algorithm to avoid this situation, however it could be that a lower interval size would permit the ETS algorithm to decrease costs through more frequent synchronization. Further research into this area is encouraged.

Next, it is possible that a single threshold value is insufficient for reducing the cost of synchronization. It may be that a better algorithm could be constructed using the principals of the ETS algorithm, but where each day is broken into zones with different threshold values. These zones may have fixed start times and durations, or they may have variable start times and durations based on historical observations of the users patterns.

Finally, while section 5.2.2 contains numerous graphs comparing possible threshold values to expected cost (5.3, 5.4, 5.5, 5.6, and 5.7), it should be noted that no such graph has been created for any of the user profiles. An understanding of how all the different threshold values compare in a specified user profile would permit a better analysis of design needs for such a protocol, and would be of benefit to this research. A fixed simulated day could be generated and re-run with a variety of possible threshold values to generate this graph, however the reason why it was not undertaken for this thesis is that the computational complexity involved in adequately computing the values for such a graph. As described in equation 5.3, the initial threshold value is set to $8 \cdot 10^{14}$. Even accounting only for positive integer threshold values, computing this many values is a significant computational under-

taking, requiring a distributed grid of computers to calculate the possibilities within reasonable time constraints. As a guide, on an Intel Core 2 Duo running at 2.16Ghz, the 250 year simulations took roughly 5 hours to complete, or approximately 18 250 simulated days per hour. At this rate, $8 \cdot 10^{14}$ simulations would take over 43 million hours to compute. Distribution over a sufficiently large array of systems, along with only evaluating the threshold values at fixed intervals would permit the creation of such a graph in reasonable time, and is suggested as a very useful avenue of further research.

8.2 Concluding Remarks

Mobile devices continue to proliferate in our society. During the writing of this thesis, Apple Inc. released their second-generation mobile telephony device, the iPhone 3G. Their competition likewise has not stood still, and both businesses and consumers have been lining up to purchase these newer devices, which emphasize higher speed data communication features and more local storage alongside the traditional voice and local directory features. The possibilities for the types of mobile applications that can be used are likewise expanding with better hardware and more ubiquitous and higher speed network connectivity. Unfortunately, at the same time data prices over cellular networks continue to be excessively expensive, particularly here in Canada, causing many businesses and consumers to try to find ways of minimizing their monthly data transfer to reduce their cellular service bills. This reduces the utility of these devices, and acts as an economic damper against the kinds of mobile applications which could be developed. It is for this reality that the ETS algorithm has been designed; by emphasizing overall cost over simply accuracy, a balance can be achieved between both ideals.

As noted in section 8.1, there is further work to be done in this area, which until now has been under researched. It is the authors hope that this work will be the precursor to further study, refinement, and innovation in this realm of endeavour.

Bibliography

[1] LimeSurvey [online]. Available from: http://www.limesurvey.org),.

[2] *Concise Oxford English dictionary*. Oxford University Press, New York, 2004.

[3] CRAWDAD: A Community Resource for Archiving Wireless Data At Dartmouth [online]. 2008. Available from: http://crawdad.cs.dartmouth.edu.

[4] ACCESS Inc. Garnet OS Documentation [online]. Available from: http://www.accessdevnet.com/index.php/20080206164/Garnet-OS/Garnet-OS.html.

[5] Acklam, P. J. An algorithm for computing the inverse normal cumulative distribution function, 2004. Available from: http://home.online.no/~pjacklam/notes/invnorm/ [cited 2-April-2008].

[6] Agarwal, S., Starobinski, D., and Trachtenberg, A. On the scalability of data synchronization protocols for PDAs and mobile devices. *Network, IEEE 16*, 4 (2002), 22–28.

[7] Alsberg, P. A., and Day, J. D. A principle for resilient sharing of distributed resources. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering* (Los Alamitos, CA, USA, 1976), IEEE Computer Society Press, pp. 562–570.

[8] Apple Inc. iSync [online]. Available from: http://www.apple.com/macosx/features/isync/.

[9] Apple Inc. Introduction to the Objective-C 2.0 Programming Language [online]. February 5 2008. Available from: `http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/`.

[10] Balachandran, A., Voelker, G. M., Bahl, P., and Rangan, P. V. Characterizing user behavior and network performance in a public wireless LAN. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2002), ACM, pp. 195–205.

[11] Barclay, B. The jSyncManager Project [online]. Available from: `http://www.jsyncmanager.org`.

[12] Barclay, B. The jSyncManager: A synchronization solution for PalmOS devices. In *Proceedings of the Wrox Wireless Developers Conference* (June 2000).

[13] Barclay, B., Price, M., and Weber-Jahnke, J. Open Technology Assisted Practice Application Suite. *ITCH 2007 Conference Proceedings* (2007).

[14] Barghouti, N. S., and Kaiser, G. E. Concurrency control in advanced database applications. *ACM Computing Surveys (CSUR) 23*, 3 (1991), 269–317.

[15] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 7 (1970), 422–426. ID: 17.

[16] Cho, J., and Garcia-Molina, H. Synchronizing a database to improve freshness. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2000), ACM, pp. 117–128.

[17] Collins-Sussmann, B., Fitzpatrick, B. W., and Pilato, C. M. *Version Control with Subversion*. O'Reilly, 2004.

[18] Cristian, F. Probabilistic clock synchronization. *Distributed Computing 3*, 3 (1989), 146–158.

[19] Dolev, S., and Welch, J. L. Self-stabilizing clock synchronization in the presence of byzantine faults. *J. ACM 51*, 5 (2004), 780–799.

[20] Doyle, M. *The A-Z of non-sexist language*. Womens Press, London, 1995.

[21] Ekenstam, T., Matheny, C., Reiher, P., and Popek, G. J. The bengal database replication system. *Distributed and Parallel Databases 9*, 3 (2001), 187–210.

[22] Foley, S. N. Conduit cascades and secure synchronization. *Proceedings of the 2000 workshop on New security paradigms* (2001), 141–150.

[23] Glasser, A. L. The evolution of a source code control system. In *Proceedings of the software quality assurance workshop on Functional and performance issues* (1978), pp. 122–125.

[24] Gottfried, B. S. *Elements of stochastic process simulation*. Prentice-Hall, Englewood Cliffs, N.J., 1984.

[25] Hansen, J. S., Reich, T., and Andersen, B. Semi-connected TCP/IP in a mobile computing environment. *The International Workshop for Information Visualization and Mobile Computing (IMC96)* (1996).

[26] Hara, T., and Madria, S. Data replication for improving data accessibility in ad hoc networks. *IEEE Transactions on Mobile Computing 5*, 11 (Nov. 2006), 1515–1532.

[27] Heckel, P. A technique for isolating differences between files. *Commun. ACM 21*, 4 (1978), 264–268.

[28] Hunt, J. W., and McIlroy, M. D. An algorithm for differential file comparison. *CSTR*, 41 (1976).

[29] Karpovsky, M., Levitin, L., and Trachtenberg, A. Data verification and reconciliation with generalized error-control codes. *IEEE Transactions on Information Theory 49*, 7 (2003), 1788–1793.

[30] Keleher, P. J. Decentralized replicated-object protocols. In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1999), ACM, pp. 143–151.

[31] Kermarrec, A.-M., Rowstron, A., Shapiro, M., and Druschel, P. The icecube approach to the reconciliation of divergent replicas. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 2001), ACM, pp. 210–218.

[32] Kheir, N. A. *Systems modelling and computer simulation*, vol. 46. M. Dekker, New York, 1988.

[33] Kistler, J. J., and Satyanarayanan, M. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems 10*, 1 (1992), 3–25. ID: 9.

[34] Knuth, D. E. *The art of computer programming.* Addison-Wesley, Reading, Mass., 1998 1997.

[35] Kotz, D., and Essien, K. Analysis of a campus-wide wireless network. *Wireless Networks 11*, 1-2 (2005), 115–133.

[36] Kuenning, G. Mersenne twist [online]. 2002. Available from: `http://www.cs.hmc.edu/~geoff/mtwist.html`.

[37] Kumar, A., and Stonebraker, M. Semantics based transaction management techniques for replicated data. In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 1988), ACM, pp. 117–125.

[38] Ladin, R., Liskov, B., Shrira, L., and Ghemawat, S. Providing high availability using lazy replication. *ACM Trans. Comput. Syst. 10*, 4 (1992), 360–391.

[39] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (1978), 558–565.

[40] Lawrence, N. D., Rowstron, A. I. T., Bishop, C. M., and Taylor, M. J. Optimising synchronisation times for mobile devices. *Advances in Neural Information Processing Systems 14* (2002).

[41] Lu, Q., and Satyanarayanan, M. Improving data consistency in mobile computing using isolation-only transactons. Tech. rep., Pittsburgh, PA, USA, 1995.

[42] Maggio, R. *The nonsexist word finder : a dictionary of gender-free usage.* Beacon Press, Boston, 1989.

[43] Matsumoto, M., and Nishimura, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS) 8*, 1 (1998), 3–30.

[44] Minsky, Y., Trachtenberg, A., and Zippel, R. Set reconciliation with nearly optimal communication complexity. *Information Theory, IEEE Transactions on 49*, 9 (2003), 2213–2218.

[45] Molli, P., Oster, G., Skaf-Molli, H., and Imine, A. Using the transformational approach to build a safe and generic data synchronizer. *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work* (2003), 212–220.

[46] Pedone, F. Boosting system performance with optimistic distributed protocols. *Computer 34*, 12 (Dec 2001), 80–86.

[47] Phan, T., Huang, L., and Dulan, C. Challenge:: integrating mobile wireless devices into the computational grid. *Proceedings of the 8th annual international conference on Mobile computing and networking* (2002), 271–278. ID: 16.

[48] Price, D. R. CVS - Concurrent Versions System [online]. 2006. Available from: `http://www.nongnu.org/cvs/`.

[49] Ratner, D., Reiher, P., and Popek, G. J. Roam: a scalable replication system for mobility. *Mob. Netw. Appl. 9*, 5 (2004), 537–544.

[50] Rochkind, M. The Source Code Control System. In *Proceedings of the 1st National Conference on Software Engineering*, pp. 37–43.

[51] Saito, Y., and Shapiro, M. Optimistic replication. *ACM Computing Surveys (CSUR) 37*, 1 (2005), 42–81.

[52] Siegemund, F., Floerkemeier, C., and Vogt, H. The value of handhelds in smart environments. *Personal and Ubiquitous Computing 9*, 2 (2005), 69–80. ID: 2.

[53] Starobinski, D., Trachtenberg, A., and Agarwal, S. Efficient PDA synchronization. *Mobile Computing, IEEE Transactions on 2*, 1 (2003), 40–51.

[54] Sudame, P., and Badrinath, B. On providing support for protocol adaptation in mobile wireless networks. *Mobile Networks and Applications 6*, 1 (2001), 43–55.

[55] Tanenbaum, A. S. *Computer Networks.* Prentice Hall PTR, 2002.

[56] Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., and Hauser, C. H. Managing update conflicts in Bayou, a weakly connected replicated storage system. *Proceedings of the fifteenth ACM symposium on Operating systems principles* (1995), 172–182.

[57] Trachtenberg, A., Starobinski, D., and Agarwal, S. Fast PDA synchronization using characteristic polynomial interpolation. *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE 3* (2002), 1510–1519 vol.3.

[58] VanDenKerkhof, E. G., Goldstein, D. H., Lane, J., Rimmer, M. J., and Dijk, J. P. V. Using a personal digital assistant enhances gathering of patient data on an acute pain management service: a pilot study. *Canadian Journal of Anesthesia 50*, 4 (2003), 368. ID: 1.

[59] Willinger, W., and Paxson, V. Where Mathematics Meets the Internet. *Notices-American Mathematical Society 45* (1998), 961–970.

[60] yang Wu, S., and Chang, Y.-T. A user-centered approach to active replica management in mobile environments. *Mobile Computing, IEEE Transactions on 5*, 11 (Nov. 2006), 1606–1619.

[61] Yu, H., and Vahdat, A. The costs and limits of availability for replicated services. *ACM Trans. Comput. Syst. 24*, 1 (2006), 70–113.

Appendix A

Simulation Library Sources

Source code for this project is available online from the following URL:

`http://sourceforge.net/projects/tricklesync`

All code developed for this project was implemented in Apple Objective-C 2.0 [9].

Appendix B

Glossary of Algorithms, Equations, Symbols, and Variables

The following is a glossary of algorithms, equations, symbols, and variables used in this thesis. The first section where they are mentioned in is listed in brackets after each description, along with the equation number (where appropriate).

Δt the absolute magnitude between two different tick intervals (5.2)

μ the mean value for a distribution (4.3.2)

σ the variance for a distribution (4.3.2)

φ_{μ,σ^2} the normal distribution with mean μ, and variance σ (4.3.2, eqn. 4.1)

$C(r_i) < P(r_i) \cdot (V(r_{i_s}) - V(r_{i_h})) \cdot k$ the inequality used to set the decision variable x_i. When this inequality holds true, $x_i = 1$, otherwise $x_i = 0$ 5.18, eqn. 5.19)

$C(R)$ a function which calculates the cost of transferring record R on the currently selected network (5.4)

d a value which denotes a device, where $d \in \{s, h\}$ (5.4)

E_{hour} the expected number of events per hour for an event E (4.3.3)

E_{max} the maximum number of permitted events per interval (4.3.4)

$G(x)$ the ETS function evaluated to determine whether to synchronize during the current tick or not (5.2, eqn. 5.1)

$G(I(t)) > T$ the ETS threshold inequality, which compares the results of function G to the threshold value T (5.2, eqn. 5.2)

$I(t)$ a function which returns a fixed-sized interval of the day, which contains tick value t (5.2)

k the estimated average cost of using an out-of-date record on the mobile device (5.2.1)

L_{entry} the time (in seconds since midnight) when the user enters locale L (4.2.3)

$L_{expectedaccesses}$ the expected number of record accesses per hour when in local L (4.2.3)

$L_{syncfrequency}$ the frequency in which the user synchronizes their mobile device when in locale L (4.2.3)

$\lim_{N_{cost_{current}}} G(x) = \infty$ the upper limit of $G(x)$ as $N_{cost_{current}}$ approaches zero (5.2.2, eqn. 5.17)

M the mobile device database corresponding to S (4.2.5)

M_k the record with an ID value of k, as stored in the mobile devices database (4.2.4)

N_{rate} the transfer rate of a network N, expressed in bytes-per-second (4.2.1)

N_{cost} the cost of transferring data on a network N, expressed in dollars per byte (4.2.1)

$N_{cost_{current}}$ the cost of transmitting a single byte on the currently selected network (5.2)

P_{access} a function which calculates the probability that a record will be accessed by the user (5.4)

$P_{bernoulli}(1) = \frac{E_{hour}}{3600}$ the Bernoulli probability of an event happening during a single tick, based on the total expected events per hour (4.3.3, eqn. 4.2)

$P_{connectionlost}(N)$ the probability that a loss of connection occurs in a one-second interval for network N (4.2.1)

$P(n) > Z$ an inequality testing whether n events have occurred during the current tick, based on the random variate Z (4.3.4, eqn. 4.4)

$P_{poisson}(n) = \sum_{k=0}^{n} \frac{(rt)^k}{k!} e^{-rt}$ the probability that n events occur during interval t, with arrival rate r (4.3.4, eqn. 4.3)

Q a set of records on the mobile device which are out-of-date, and thus requiring synchronization (5.4)

$|Q|$ the number of records in Q (5.4)

r_i a record in set Q, requiring synchronization, with index value i (5.4)

R_{ID} the unique identification number of record R (4.2.4)

R_{size} the size, expressed in kilobytes, of record R (4.2.4)

$R_{revision}$ the modification revision of record R, expressed as a positive integer (4.2.4)

S the remote server database (4.2.5)

S_k the record with an ID value of k, as stored in the remote server database (4.2.4)

$S_{maxarrivals}$ the maximum simultaneous record modifications permitted by the simulation in one tick in database S(4.2.5)

S_{max} the largest record ID number in database S (4.2.5)

t a specific tick value during the day.

T a threshold value, as used by the ETS algorithm (5.2)

T_{init} the initial threshold value for the ETS algorithm, equal to $8 \cdot 10^{14}$ (5.2.1, eqn. 5.3)

T_{lower} the lower value of the current ETS algorithm threshold range (5.2.1)

$T_{lower_{init}} = \frac{T_{midpoint} - \frac{900}{max(N_{cost_{current}})}}{2}$ the threshold lower bound calculated after the first day of using the ETS algorithm (5.2.1, eqn. 5.6)

$T_{lower_{new}} = T_{lower}$ the new value for T_{lower} when T_{lower} is found to have the best cost reducing benefit (5.2.1, eqn. 5.12)

$T_{lower_{new}} = T_{lower} + \frac{T_{upper} - T_{lower}}{10}$ the new value for T_{lower} when $T_{midpoint}$ is found to have the best cost reducing benefit (5.2.1, eqn. **??**)

$T_{lower_{new}} = T_{midpoint}$ the new value for T_{lower} when T_{upper} is found to have the best cost reducing benefit (5.2.1, eqn. 5.8)

$T_{midpoint}$ the mid-point of the current ETS algorithm threshold range (5.2.1)

$T_{midpoint_{init}} = \frac{3600}{max(N_{cost_{current}})}$ the threshold midpoint calculated after the first day of using the ETS algorithm (5.2.1, eqn. 5.4)

$T_{midpoint_{new}} = T_{midpoint}$ the new value for $T_{midpoint}$ when $T_{midpoint}$ is found to have the best cost reducing benefit (5.2.1, eqn. 5.14)

$T_{midpoint_{new}} = \frac{T_{midpoint} - T_{lower}}{2}$ the new value for $T_{midpoint}$ when T_{lower} is found to have the best cost reducing benefit (5.2.1, eqn. 5.13)

$T_{midpoint_{new}} = \frac{T_{upper} - T_{midpoint}}{2}$ the new value for $T_{midpoint}$ when T_{upper} is found to have the best cost reducing benefit (5.2.1, eqn. 5.10)

T_{upper} the upper value of the current ETS algorithm threshold range (5.2.1)

$T_{upper_{init}} = \frac{\frac{43200}{max(N_{cost_{current}})}}{2} - T_{midpoint}$ the threshold upper bound calculated after the first day of using the ETS algorithm (5.2.1, eqn.

$T_{upper_{new}} = T_{midpoint}$ the new value for T_{upper} when T_{lower} is found to have the best cost reducing benefit (5.2.1, eqn. 5.11)

$T_{upper_{new}} = T_{upper}$ the new value for T_{upper} when T_{upper} is found to have the best cost reducing benefit (5.2.1, eqn. 5.9)

$T_{upper_{new}} = T_{upper} - \frac{T_{upper} - T_{lower}}{10}$ the new value for T_{upper} when $T_{midpoint}$ is found to have the best cost reducing benefit (5.2.1, eqn. 5.15)

$T_{upper} - T_{lower} > \epsilon$ an inequality used to determine if T_{upper} and T_{lower} are within a small bounds value ϵ of each other (5.2.1, eqn. 5.7) 5.5)

U the user object for the current simulation run (4.2.6)

$y = mx + b$ the standard linear equation, where m is the slope of the line, and b is the y-intersection; when used to describe the cost of synchronization, m is the intangible cost of using out-of-date records, while b is the cost of using network resources (4.4.2, eqn. 4.5)

$V(R, d)$ a function which calculates the version number of a record R on device d (5.4)

x_i a decision variable which decides whether record r_i is to be synchronized, where $x_i \in \{0, 1\}$ (5.4)

$XC = \sum_{i=1}^{n} C(r_i)x_i + P_{access}(r_i)\overline{x_i} \cdot (V(r_i, s) - V(r_i, h)) \cdot k$ the expected cost of synchronizing the set Q at a specific tick (5.4, eqn. 5.18)

Z a uniformly distributed random number, as selected from a PRNG (4.3.4)

Appendix C

Survey Invitation to Participate

(Sent to invitees via e-mail).

You are invited to participate in a study entitled Real-World Mobile Device Synchronization Frequency in Commercial Applications that is being conducted by Brad BARCLAY.

Brad Barclay is a Graduate Student in the Department of Computer Science at the University of Victoria. As a graduate student, I am required to conduct research as part of the requirements for a degree in Computer Science.

The purpose of this study is to determine the norms and best practices used by organizations when synchronizing handheld computing devices (mobile phones, Palm-style handheld computers, etc.) to corporate databases. This research is important; the lack of existing public research in this area prevents the comparison of different practices, both real and hypothesized, in an attempt to improve overall cost-effectiveness of data synchronization for mobile devices.

The survey consists of seven questions, and will take no more than five minutes to complete. No identifying information of any type will be requested, stored, or retained, including the names of participants, organizations they belong to, IP addresses, or digital signatures.

To participate, you are invited to visit the the survey website at [WEB ADDRESS OF SURVEY SITE]. You are invited to take a moment to read the Letter of Information for Implied Consent, and at your option to complete the short survey.

Please note that participation in this study is VOLUNTARY. Invitees should not feel obligated to participate based on professional affiliation.

If you have any questions pertaining to this survey, you are invited to contact the researcher, Brad BARCLAY, via e-mail at <yaztromo@cs.uvic.ca>.

Your time and consideration are greatly appreciated.

Brad BARCLAY

M.Sc. Candidate,

Department of Computer Science,

University of Victoria,

Victoria, BC, CANADA

Appendix D

Letter of Implied Consent

Optimizing Synchronization Cost for Mobile Devices: The Expedient Trickle Sync Algorithm

You are invited to participate in a study entitled Real-World Mobile Device Synchronization Frequency in Commercial Applications that is being conducted by Brad BARCLAY.

Brad Barclay is a Graduate Student in the Department of Computer Science at the University of Victoria and you may contact him if you have further questions via e-mail at yaztromo@uvic.ca.

As a graduate student, I am required to conduct research as part of the requirements for a degree in Computer Science. It is being conducted under the supervision of Dr. Jens H. WEBER. You may contact my supervisor at jens@uvic.ca.

Purpose and Objectives

The purpose of this research project is to determine the norms and best practices used by organizations when synchronizing handheld computing devices (mobile phones, Palm-style handheld computers, etc.) to corporate databases.

Importance of this Research

Research of this type is important because such information is currently lacking, and can be of great use to researchers who are attempting to find better and more cost-effective mechanisms for conducting the synchronization of handheld computing devices.

Participants Selection

You are being asked to participate in this study because you are/were a known user of the jSyncManager, a Free Open Source software package for synchronizing PalmOS-based handheld devices.

What is involved

If you agree to voluntarily participate in this research, your participation will include answering a simple 5-minute web-based survey. The questions in the survey may, at your option, require a brief analysis of your existing databases and/or log files used or generated when synchronizing mobile devices within your organization.

Inconvenience

Participation in this study may cause some inconvenience to you, including the brief amount of time required to respond to this survey.

Risks

There are no known or anticipated risks to you by participating in this research.

Benefits

The potential benefits of your participation in this research include more accurate models on how existing organizations handle the process of synchronizing information from portable, semi-connected devices, which will allow the researcher to evaluate more cost-effective protocols and best practices for performing such synchronization tasks in the future.

Voluntary Participation

Your participation in this research must be completely voluntary. If you do decide to participate, you may withdraw at any time without any consequences or any explanation. If you do withdraw from the study your data will be impossible to remove once submitted, as no records of what data is attached to which respondent will be retained.

Anonymity

In terms of protecting your anonymity no identifying information will be requested or permitted. No digitally identifying information (such as MAC addresses or IP addresses) will be recorded, stored, or retained as part of the survey submission process.

Confidentiality

Your confidentiality and the confidentiality of the data will be protected by the lack of any identifying information being requested, stored, or retained. All information will be stored within an encrypted database on a machine stored in a locked office within the University of Victorias Computer Science Department.

Dissemination of Results

It is anticipated that the results of this study will be shared with others in the following ways: via published articles, thesis, and via presentations. The final study results will be made available to participants upon request.

Disposal of Data

Data from this study will not be disposed of. The anonymous survey responses will be recorded in an appendix of the researchers thesis. The electronic database used to collect and store the initial results will be deleted using a 7-pass erase function at the completion of this study.

Contacts

Individuals that may be contacted regarding this study include Mr. Brad BARCLAY <yaztromo@cs.uvic.ca> and Dr. Jens H. WEBER <jens@uvic.ca>.

In addition, you may verify the ethical approval of this study, or raise any concerns you might have, by contacting the Human Research Ethics Office at the University of Victoria (250-472-4545 or ethics@uvic.ca).

By completing and submitting the questionnaire, YOUR FREE AND INFORMED CONSENT IS IMPLIED and indicates that you understand the above conditions of participation in this study and that you have had the opportunity to have your questions answered by the researchers.

Please retain a copy of this letter for your reference.

Appendix E

User Model Study Questionnaire

1. During regular business hours, how many times on average do you expect users to synchronize their mobile devices against your organizations database(s)?

2. Considering your answer to question 1, which of the following best describes how you arrived at your response? (check one):

   - Organization mandated best practice
   - Result from computerized synchronization logs or database query
   - Approximate guess based on experience with users

3. During non-business hours (evenings, weekends, holidays, etc.) how many times per day do you expect users to synchronize their mobile devices against your organizations database(s)?

4. Considering your answer to question 2, which of the following best describes how you arrived at your response? (check one):

   - Organization mandated best practice
   - Result from computerized synchronization logs or database query
   - Approximate guess based on experience with users

5. Approximately how many records are contained within the database table(s) your users synchronize their mobile devices against?

6. During regular business hours, how many times on average are the records within this(/these) database table(s) expected to be modified (either through the synchronization of modified records, or through other means) per day?

7. Considering your answer to question 6, which of the following best describes how you arrived at your response? (check one):

    - Result from computerized synchronization logs or database query
    - Approximate guess based on experience with the database(s) in question.

Appendix F

Simulation XML Property Files

This appendix lists the XML property files used to evaluate the ETS algorithm for each of the modelled users. Versions of these users were created for each of the synchronization algorithms evaluated. Copies of these profiles can be found in the ETS Subversion repository, as detailed in Appendix A.

F.1 Simple User A

```
<?xml version="1.0" encoding="UTF-8"?>
<etssimulation>
   <networks>
      <network name="WiFi" cost="0.000000000001" xferrate="49000.0" expecteddisocnnectsperhour="0.01"/>
   </networks>

   <locations>
      <location name="home">
         <localnet>WiFi</localnet>
      </location>
   </locations>

   <userschedule>
      <entry locname="home" time="0" expectedsyncs="0.166666666666667" expectedaccesses="5"/>
   </userschedule>

   <database numrecords="5000" arrivalrate="50.0" interval="3600" maxarrivals="10"/>

   <syncprotocol name="ETSAdaptor" k="5"/>
</etssimulation>
```

F.2 Simple User B

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<etssimulation>
   <networks>
      <network name="GPRS" cost="0.000048828125" xferrate="57.0" expecteddisocnnectsperhour="0.1"/>
   </networks>

   <locations>
      <location name="home">
         <localnet>GPRS</localnet>
      </location>
   </locations>

   <userschedule>
      <entry locname="home" time="0" expectedsyncs="0.166666666666667"" expectedaccesses="5"/>
   </userschedule>

   <database numrecords="5000" arrivalrate="50.0" interval="3600" maxarrivals="10"/>

   <syncprotocol name="ETSAdaptor" k="5"/>
</etssimulation>
```

F.3 Business User A, C

```
<?xml version="1.0" encoding="UTF-8"?>
<etssimulation>
   <networks>
      <network name="GPRS" cost="0.000048828125" xferrate="57.0" expecteddisocnnectsperhour="0.1"/>
      <network name="WiFi" cost="0.000000000001" xferrate="49000.0" expecteddisocnnectsperhour="0.01"/>
      <network name="CommercialWiFi" cost="0.000001" xferrate="49000" expecteddisocnnectsperhour="0.01"/>
   </networks>

   <locations>
      <location name="home">
         <localnet>GPRS</localnet>
         <localnet>WiFi</localnet>
      </location>
      <location name="car">
         <localnet>GPRS</localnet>
      </location>
      <location name="office">
         <localnet>GPRS</localnet>
         <localnet>WiFi</localnet>
      </location>
      <location name="lunch">
         <localnet>GPRS</localnet>
         <localnet>CommercialWiFi</localnet>
      </location>
   </locations>

   <userschedule>
      <entry locname="home" time="0" expectedsyncs="0.0" expectedaccesses="0.00001"/>
      <entry locname="home" time="25200" expectedsyncs="0.1" expectedaccesses="0.2"/>
      <entry locname="car" time="28800" expectedsyncs="0.0" expectedaccesses="0.00001"/>
      <entry locname="office" time="32400" expectedsyncs="0.25" expectedaccesses="10.0"/>
      <entry locname="lunch" time="43200" expectedsyncs="0.1" expectedaccesses="2.0"/>
```

```
        <entry locname="office" time="46800" expectedsyncs="0.25" expectedaccesses="10.0"/>
        <entry locname="car" time="61200" expectedsyncs="0.0" expectedaccesses="0.00001"/>
        <entry locname="home" time="64800" expectedsyncs="0.1" expectedaccesses="0.5"/>
        <entry locname="home" time="79200" expectedsyncs="0.0" expectedaccesses="0.00001"/>
    </userschedule>

    <database numrecords="5000" arrivalrate="40.0" interval="3600" maxarrivals="10"/>

    <syncprotocol name="ETSAdaptor" k="5"/>
</etssimulation>
```

F.4 Business User B

```
<?xml version="1.0" encoding="UTF-8"?>
<etssimulation>
    <networks>
        <network name="GPRS" cost="0.000048828125" xferrate="57.0" expecteddisocnnectsperhour="0.1"/>
        <network name="WiFi" cost="0.000000000001" xferrate="49000.0" expecteddisocnnectsperhour="0.01"/>
        <network name="CommercialWiFi" cost="0.000001" xferrate="49000" expecteddisocnnectsperhour="0.01"/>
    </networks>

    <locations>
        <location name="home">
            <localnet>GPRS</localnet>
            <localnet>WiFi</localnet>
        </location>
        <location name="car">
            <localnet>GPRS</localnet>
        </location>
        <location name="office">
            <localnet>GPRS</localnet>
            <localnet>WiFi</localnet>
        </location>
        <location name="lunch">
            <localnet>GPRS</localnet>
            <localnet>CommercialWiFi</localnet>
        </location>
    </locations>

    <userschedule>
        <entry locname="home" time="0" expectedsyncs="1.0" expectedaccesses="0.00001"/>
        <entry locname="home" time="25200" expectedsyncs="1.0" expectedaccesses="0.2"/>
        <entry locname="car" time="28800" expectedsyncs="1.0" expectedaccesses="0.00001"/>
        <entry locname="office" time="32400" expectedsyncs="1.0" expectedaccesses="10.0"/>
        <entry locname="lunch" time="43200" expectedsyncs="1.0" expectedaccesses="2.0"/>
        <entry locname="office" time="46800" expectedsyncs="1.0" expectedaccesses="10.0"/>
        <entry locname="car" time="61200" expectedsyncs="1.0" expectedaccesses="0.00001"/>
        <entry locname="home" time="64800" expectedsyncs="1.0" expectedaccesses="0.5"/>
        <entry locname="home" time="79200" expectedsyncs="1.0" expectedaccesses="0.00001"/>
    </userschedule>

    <database numrecords="5000" arrivalrate="40.0" interval="3600" maxarrivals="10"/>

    <syncprotocol name="FastSyncAdaptor"/>
```

```
</etssimulation>
```

F.5 Mobile Salesperson A

```
<?xml version="1.0" encoding="UTF-8"?>
<etssimulation>
   <networks>
      <network name="GPRS" cost="0.000048828125" xferrate="57.0" expecteddisocnnectsperhour="0.1"/>
      <network name="WiFi" cost="0.000000000001" xferrate="49000.0" expecteddisocnnectsperhour="0.01"/>
   </networks>

   <locations>
      <location name="home">
         <localnet>GPRS</localnet>
         <localnet>WiFi</localnet>
      </location>
      <location name="car">
         <localnet>GPRS</localnet>
      </location>
      <location name="office">
         <localnet>GPRS</localnet>
         <localnet>WiFi</localnet>
      </location>
      <location name="CustomerSite">
         <localnet>GPRS</localnet>
      </location>
   </locations>

   <userschedule>
      <entry locname="home" time="0" expectedsyncs="0.0" expectedaccesses="0.00001"/>
      <entry locname="home" time="21600" expectedsyncs="4" expectedaccesses="4"/>
      <entry locname="car" time="22500" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="office" time="24900" expectedsyncs="4" expectedaccesses="3"/>
      <entry locname="CustomerSite" time="27000" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="27600" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="CustomerSite" time="27900" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="29700" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="CustomerSite" time="30000" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="32400" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="home" time="34200" expectedsyncs="4" expectedaccesses="4"/>
      <entry locname="car" time="34800" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="CustomerSite" time="35700" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="37800" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="CustomerSite" time="37920" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="41400" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="office" time="44100" expectedsyncs="4" expectedaccesses="3"/>
      <entry locname="car" time="46800" expectedsyncs="4" expectedaccesses="5"/>

      <entry locname="CustomerSite" time="48600" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="49500" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="CustomerSite" time="49620" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="50700" expectedsyncs="4" expectedaccesses="5"/>
      <entry locname="CustomerSite" time="50820" expectedsyncs="4" expectedaccesses="2"/>
      <entry locname="car" time="51900" expectedsyncs="4" expectedaccesses="5"/>
```

```
        <entry locname="home" time="53100" expectedsyncs="4" expectedaccesses="5"/>
        <entry locname="home" time="63000" expectedsyncs="4" expectedaccesses="5"/>
        <entry locname="home" time="82800" expectedsyncs="0.0" expectedaccesses="0.00001"/>
    </userschedule>

    <database numrecords="1000" arrivalrate="20.0" interval="3600" maxarrivals="10"/>

    <syncprotocol name="ETSAdaptor" k="10">
        <!-- ...protocol specific nodes go here... -->
    </syncprotocol>
</etssimulation>
```

F.6 Truck Driver A

```
<?xml version="1.0" encoding="UTF-8"?>
<etssimulation>
    <networks>
        <network name="GPRS" cost="0.000048828125" xferrate="57.0" expecteddisocnnectsperhour="0.1"/>
        <network name="WiFi" cost="0.000000000001" xferrate="49000.0" expecteddisocnnectsperhour="0.01"/>
        <network name="CommercialWiFi" cost="0.000001" xferrate="49000" expecteddisocnnectsperhour="0.01"/>
    </networks>

    <locations>
        <location name="road">
            <localnet>GPRS</localnet>
        </location>
        <location name="office">
            <localnet>GPRS</localnet>
            <localnet>WiFi</localnet>
        </location>
        <location name="truckstop">
            <localnet>GPRS</localnet>
            <localnet>CommercialWiFi</localnet>
        </location>
    </locations>

    <userschedule>
        <entry locname="office" time="0" expectedsyncs="1.0" expectedaccesses="0.00001"/>
        <entry locname="road" time="32400" expectedsyncs="1.0" expectedaccesses="10.0"/>
        <entry locname="truckstop" time="43200" expectedsyncs="1.0" expectedaccesses="1.0"/>
        <entry locname="road" time="46800" expectedsyncs="1.0" expectedaccesses="10.0"/>
        <entry locname="office" time="61200" expectedsyncs="1.0" expectedaccesses="0.00001"/>
    </userschedule>

    <database numrecords="2000" arrivalrate="20.0" interval="3600" maxarrivals="5"/>

    <syncprotocol name="ETSAdaptor" k="250"/>
</etssimulation>
```

Appendix G

Raw Result Data

G.1 User Model Protocol Comparisons

The following tables contains the raw result data for the user model simulations runs, with evaluations for each of the five tested synchronization algorithms. Each simulation ran for 365 simulated days, and was repeated 250 times.

Protocol	Cost	Variance	Data (KiB/day)
Null Sync	$y = 5254.741x + 0.000$	$y = 9521520.432x^2 + 0.000x + 0.000$	0.000
Random Sync	$y = 15.427x + 0.0001$	$y = 141.077x^2 - 0.001x + 0.000$	100573.916
Slow Sync	$y = 7.217x + 0.0002$	$y = 28.201x^2 - 0.001x + 0.000$	212202.098
Fast Sync	$y = 7.1973x + 0.00001$	$y = 27.842x^2 - 0.000x + 0.000$	11357.976
ETS	$y = 1.828x + 0.00001$	$y = 64.730x^2 - 0.000x + 0.000$	11974.357

Table G.1: Raw Results for Simple User A

Protocol	Cost	Variance	Data (KiB/day)
Null Sync	$y = 5258.373x + 0.000$	$y = 9531964.491x^2 + 0.000x + 0.000$	0.000
Random Sync	$y = 15.508x + 4980.831$	$y = 138.112x^2 - 26964.959x + 8289616.021$	99616.623
Slow Sync	$y = 7.280x + 10503.982$	$y = 28.545x^2 - 27592.001x + 25761231.733$	210079.646
Fast Sync	$y = 7.201x + 568.161$	$y = 27.732x^2 - 135.601x + 36435.194$	11363.225
ETS	$y = 4.206x + 586.705$	$y = 79.149x^2 - 58.678x + 7138.765$	11734.091

Table G.2: Raw Results for Simple User B

Protocol	Cost	Variance	Data (KiB/day)
Null Sync	$y = 2601.496x + 0.000$	$y = 2379303.396x^2 + 0.000x + 0.000$	0.000
Random Sync	$y = 12.826x + 2927.926$	$y = 98.856x^2 - 17261.703x + 4881234.606$	58558.527
Slow Sync	$y = 6.106x + 5.534$	$y = 33.102x^2 - 22.533x + 307.516$	126908.148
Fast Sync	$y = 6.090x + 0.388$	$y = 32.592x^2 - 0.073x + 3.109$	8834.313
ETS	$y = 2.631x + 16.581$	$y = 445.026x^2 - 67.173x + 98.227$	9463.159

Table G.3: Raw Results for Business User A

Protocol	Cost	Variance	Data (KiB/day)
Null Sync	$y = 2601.320x + 0.000$	$y = 2381818.359x^2 + 0.000x + 0.000$	0.000
Random Sync	$y = 1.699x + 29894.357$	$y = 26.540x^2 - 3684.460x + 50087437.844$	597887.143
Slow Sync	$y = 0.598x + 5056.932$	$y = 0.690x^2 - 266.979x + 12509618.704$	1212665.057
Fast Sync	$y = 0.593x + 40.253$	$y = 0.682x^2 - 0.187x + 1012.800$	9562.598
ETS	$y = 2.631x + 16.581$	$y = 445.026x^2 - 67.173x + 98.227$	9463.159

Table G.4: Raw Results for Business User B

Protocol	Cost	Variance	Data (KiB/day)
Null Sync	$y = 2600.942x + 0.000$	$y = 2379474.155x^2 + 0.000x + 0.000$	0.000
Random Sync	$y = 2386.961x + 17751.285$	$y = 2667999.811x^2 - 82392670.161x + 6348884364.353$	355025.696
Slow Sync	$y = 31.392x + 2200.284$	$y = 76425.250x^2 - 117994.612x + 49990120.503$	50699776.318
Fast Sync	$y = 12.027x + 156.655$	$y = 10938.001x^2 - 1863.027x + 509855.111$	3573580.763
ETS	$y = 2.599x + 5422.050$	$y = 385.373x^2 - 21631.832x + 19892766.652$	3855508.960

Table G.5: Raw Results for Business User C

Protocol	Cost	Variance	Data (KiB/day)
Null Sync	$y = 6200.217x + 0.000$	$y = 13529065.072x^2 + 0.000x + 0.000$	0.000
Random Sync	$y = 7.099x + 17037.782$	$y = 725.699x^2 - 949.207x + 5812643.143$	340755.648
Slow Sync	$y = 0.502x + 14129.607$	$y = 0.563x^2 - 153.821x + 7089395.892$	687157.702
Fast Sync	$y = 0.505x + 92.124$	$y = 0.564x^2 + 6.630x + 1168.446$	4693.526
ETS	$y = 7.12x + 14.04$	$y = 754.732x^2 - 14.620x + 498.798$	4747.325

Table G.6: Raw Results for Mobile Salesperson A

Protocol	Cost	Variance	Data (KiB/day)
Null Sync	$y = 3153.023x + 0.000$	$y = 3497047.492x^2 + 0.000x + 0.000$	0.000
Random Sync	$y = 2.958x + 12001.305$	$y = 91.026x^2 - 1845.995x + 8008100.491$	240026.103
Slow Sync	$y = 0.719x + 7083.067$	$y = 0.855x^2 - 1116.741x + 7033886.037$	485710.547
Fast Sync	$y = 0.721x + 69.802$	$y = 0.858x^2 + 0.635x + 347.205$	4769.964
ETS	$y = 2.338x + 36.073$	$y = 95.991x^2 - 7.154x + 831.858$	4607.182

Table G.7: Raw Results for Truck Driver A

G.2 ETS algorithm evaluations for varying values of k

The following tables contains the raw result data for the simulations runs evaluating only the ETS algorithm, but with differing input values k, where k is the expected average out-of-date record access cost. Each simulation ran for 365 simulated days, and was repeated 100 times.

k	Cost	Variance	Data (KiB/day)
0.1	$y = 7.001x + 573.082$	$y = 81.842x^2 - 9.796x + 11775.567$	11461.646
1	$y = 6.965x + 573.962$	$y = 76.013x^2 - 9.223x + 12485.071$	11479.24
10	$y = 2.709x + 594.982$	$y = 83.878x^2 - 22.592x + 3806.341$	11899.641
100	$y = 1.746x + 598.418$	$y = 68.807x^2 + 22.025x + 2291.852$	11968.361
1000	$y = 1.741x + 598.588$	$y = 66.911x^2 + 20.828x + 2251.042$	11971.762
10000	$y = 1.773x + 599.340$	$y = 72.017x^2 + 27.671x + 2051.656$	11986.808

Table G.8: Raw Results for Simple User B when evaluating the ETS algorithm at different values of k

k	Cost	Variance	Data (KiB/day)
0.1	$y = 6.788x + 1.368$	$y = 713.248x^2 - 3.142x + 72.147$	4735.662
1	$y = 7.231x + 12.842$	$y = 762.211x^2 - 16.919x + 506.862$	4751.448
10	$y = 7.254x + 13.1779$	$y = 794.299x^2 - 22.691x + 719.613$	4740.25
100	$y = 7.241x + 26.484$	$y = 794.299x^2 - 22.691x + 719.613$	4740.25
1000	$y = 7.016x + 27.914$	$y = 762.609x^2 - 21.005x + 644.540$	4749.796
10000	$y = 7.105x + 22.548$	$y = 769.260x^2 - 24.412x + 654.498$	4749.619

Table G.9: Raw Results for Mobile Salesperson A when evaluating the ETS algorithm at different values of k