Database syncrhonization between devices

A new synchronization protocol for SQLite databases

SHITIAN LONG

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Database synchronization between devices

A new synchronization protocol for SQLite databases

Shitian Long

Master thesis

2011.05.28

Examiner: G. Q. Maguire Jr.

Industrial Advisor: Anibal Wainstein, Diabetes Tools Sweden AB

School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

Today people have multiple personal computers, personal digital assistants and smart phones. Today's advanced handheld devices have powerful processors, with a process frequency of up to 1 GHz, huge storage capacities, flash storage capacities up to 32 GB, a large (multi) touch screen, and a user-friendly user interface. Additionally, the device may have various input and output devices, thus leading to people utilizing different devices for different use cases. In order to provide the latest information to the users via any of their devices, data synchronization becomes a requirement for users.

There are many data synchronization solutions for synchronizing database records and files. In the current database synchronization solutions, there is no clear source and target. For example, consider the case where a PDA synchronizes with a PC; the record could have been edited (changed) on both the PDA and PC. In this case it is not clear which should be synchronized with what should be the source for the value. In contrast, a files synchronization system has a clearly specified synchronization source and destination structure. In this case the client synchronizes their files with that of the server. In a version control system the client devices synchronize files with a repository acting as a version control server. There are many synchronization protocols and each has been designed for a different purpose. Protocols for synchronizing database records often provide continuous synchronization, leading to a lot of data being exchanging during the synchronization process, as a result the synchronization process takes a longer period of time, but maintains the semantics of the database updates (either a complete transaction completes or it must be rolled back) On the other hand, protocols for synchronizing files may require a short synchronization time, as the whole file transferred and replaces the previous version of the file at the destination. Note file synchronization may also transfer only the differences between the files, with a local transformation of an existing file copy of the by applying these differences as updates to the files. Sending only the updates to a file enables large files with a small number of changes to be quickly updated. However, file based updated does not efficiently support record level updates of a database.

In this thesis, we designed a new synchronization protocol for synchronizing two SQLite databases. This synchronization protocol borrows from (and hence offers the advantages of) a version control system in order to rapidly perform SQLite database synchronization. Moreover, this solution brings SQLite database additional functions, for example supporting multiple –user, transaction logs, and data roll-back.

Abstrakt

Idag människor har flera datorer, handdatorer och smarta telefoner. Dagens avancerade handdatorer har kraftfulla processorer, med en process frekvens av upp till 1 GHz, stora lagringskapaciteten, flash lagringskapacitet upp till 32 GB, en stor (multi) pekskärm, och ett användarvänligt användargränssnitt. Dessutom kan enheten ha olika in-och utenheter, vilket leder till människor som använder olika enheter för olika användningsområden. För att ge den senaste informationen till användarna via någon av deras produkter, blir datasynkronisering ett krav för användarna.

Det finns många datasynkronisering lösningar för synkronisering av databasen register och arkiv. I den nuvarande databaslösningar synkronisering, finns det ingen klar källa och mål. Till exempel anser de fall där en PDA synkroniseras med en PC, posten kunde ha utformats (ändrade) på både PDA och PC. I detta fall är det inte klart vilka bör synkroniseras med det som borde vara en källa för värdet. Däremot har en filer synkronisering system en tydligt angiven synkronisering källa och destination struktur. I detta fall kunden synkroniserar sina handlingar med att på servern. I en version styrsystem klientenheter synkronisera filer med ett slutförvar fungerar som en version Control Server. Det finns många synkronisering protokoll och varje har designats för ett annat ändamål. Protokoll för att synkronisera databasposter ger ofta kontinuerlig synkronisering, vilket leder till en massa data som utbyte under synkroniseringen, som ett resultat av synkroniseringen tar längre tid, men behåller semantik av databasen uppdateras (antingen ett komplett transaktion slutför eller så måste rullas tillbaka) Å andra sidan kan protokoll för att synkronisera filer kräver en kort synkronisering tid, eftersom de överförda hela filen och ersätter den tidigare versionen av filen på destinationen. Obs filsynkronisering kan också överföra endast skillnaderna mellan filerna, med en lokal omvandling av en befintlig fil kopia av genom att tillämpa dessa skillnader som uppdateringar till filerna. Endast skicka uppdateringarna till en fil gör att stora filer med ett litet antal ändringar som ska snabbt uppdateras. Men baserad fil uppdateras inte effektivt stöd till rekordnivån uppdateringar av en databas.

I denna avhandling har vi tagit en ny synkronisering protokoll för synkronisering av två SQLite-databaser. Denna synkronisering protokoll lånar från (och därmed fördelen av) ett versionshanteringssystem för att snabbt utföra SQLite databas synkronisering. Dessutom ger denna lösning SQLite databas ytterligare funktioner, till exempel stöd för flera användare, transaktionsloggar och data rulla tillbaka.

Acknowledgements

My first acknowledgment goes to Anibal Wainstein, my supervisor who guided me with regard to the standard of enterprise level distribution system programming, the magic of design patterns and how to work with a development team. I also acknowledge Mehmet Yıldız and Muhammad Ahmad, my team members at industry, thank you for discussing technical problems with me. Mehmet Yıldız especially offered me a great deal of help in C# programming with the .NET framework. I learned a lot from both of you. I thank all my colleagues, Fredrik Wallander, Markus Eriksson, and others for your kind help during my project at industry. I thank my boss Lars Liljeryd, for his support and continuous encouragement.

I thank Professor Gerald Q. "Chip" Maguire Jr. who was my supervisor at KTH. He reviewed my thesis carefully and gave me a lot of useful comments. I also thank Professor Mark Smith, who gave me much good advice.

Finally, I give special thanks to Jian Wang, my unique fellow classmate at KTH. I want to say, you are the one who always helped me with programming and I was very so lucky and enjoyed working with you.

Table of Contents

List of Figures

List of Tables

Glossary

ACID	Atomic, Consistent, Isolated, and Durable
CORBA	Common Object Request Broker Architecture
DBMS	Database Management System
DTD	Document Type Definitions
FTP	File Transfer Protocol
HSPA	High Speed Packet Access
HTTP	HyperText Transfer Protocol
HTTPS	HTTP over SSL
ICT	Information Communication Technology
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
LTE	Long Term Evolution
MS	Microsoft
PC	Personal Computer
PIM	Personal Information Message
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
SFTP	SSH File Transfer Protocol
SSL	Secure Socket Layer
UDP	User Datagram Protocol
WCDMA	Wideband Code Division Multiple Access
WSDL	Web Service Definition Language
XML	eXtensible Markup Language

1 Introduction

During the last decade people have been using more and more information communication technology (ICT) devices; based on annual reports of global computer usage, cellular phones and other consumer electronic devices ownership has increased and will continue to increase. [1] [2] [3] In some countries, the majority of people have more than one computer or cellular phone. Users store different amounts of data in many devices. These amounts of data can be stored or manipulated by a Personal Information Manager (PIM) and it may include personal contact information, calendars, schedules, or binary files such as MP3 media files and applications files. Different devices are used in different situations. For example, at home a desktop computer might be used, while on the train a cellular phone or tablet might be used often, and at the office, customer sites, and in hotel rooms, the company's laptop might be used. In any cases, users need to review and update the same type of data on a different machine. Additionally the users expect to have the latest version of their records or files, despite they use different devices in different situations. Therefore data synchronization between different devices is an increasingly important users' requirement.

In order to make sure all the users' devices have consistent information despite users making update using different devices, these devices need to send updates messages between themselves. For example, if users insert new contact record into their office laptop, then this updated contact record should propagate to all of their contact databases (i.e., the contact database on each of their different devices). If the user downloads an MP3 media file to their cellular phone, then they might want this media file propagated to their laptop and desktop computers. If the users update a document file on their home desktop and might this document concerns their work, then users may expect that their corporate laptop to receive an updated copy of this document. If this document is subjected to vision control system, then updated version might be placed in the company's document repository. In these cases, the users' devices have to send messages between themselves with the changes necessary to ensure consistent information in all of the users' devices. These changes include database changes, binary files changes, or ASCII (text)[1] files changes (see figure1).

[1] An ASCII file is defined as a file that consists of ASCII characters. It is usually created by using a text editor such as emacs, pico, vi, Notepad, etc. [43]

s

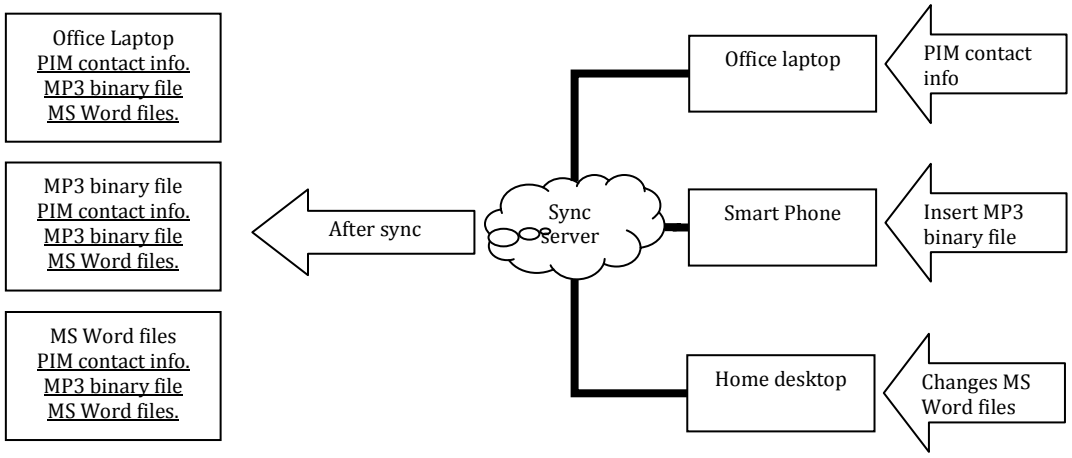

Figure 1: Overview of synchronization across three systems

In the scenario shown in the Figure 1, a user has three devices: office laptop, smart phone and home desktop. The user makes changes to databases records, binary files or other files on these different devices. In the example above, a MP3 file is used as an example of a binary file and Microsoft Word *.docx format file is used as an example of a text file[2]. After synchronization, all the user's devices should have the same contents. Note that this example assume that all devices should be consistent and do not consider policies about what updates should or should not be propagated to the other devices. The issues of such policies will be addressed in Chapter two.

There are four important issues involved when discussing synchronization between the three devices in the scenario above. First recognizing changes on end devices; for example, how do end devices' application recognize and record a user's actions on a database, like the user's inserting, deleting or modifying records actions. The next question will be how end devices encode the user's changes and send the other devices of these changes. For example, in this case shown in Figure 1 above, the changed PIM contact information could be encoded in either a proprietary binary file format or a plain text file (such as SyncML standard format text file) in order to inform a synchronization server of these changes. After the user's changes on one device has been encoded and sent to the other devices, thus the next question will be which protocol will be used to send these changes. The choices of protocols can be vary depending on the types of physical and link layer connections that are available. For example, a smart phone might be connected via a USB link with a computer, in this case the encoded change data will be sent via the USB connection using a serial link protocol [4]. In other case, the devices might be connected via an IP network; so the devices will send packetized encoded information via a TCP connection, a series of UDP datagram or via some other transport protocols. After the information about the changes is about to send to a synchronization server (we assume that there is an application on the destination devices that will make apply the changes to the local device in order to synchronize the target device with the source device), next question concerns where a

[2] In this example we consider Microsoft Word *.docx format a textual format file, since the *.docx format is based on XML.

synchronization server or servers are. Therefore our choices of mechanism to propagate the changes (i.e., to transport the change information) as well as our choice of method to apply the changes may depend upon the number of recipients. For example, if multiple recipients get the changes, then either multiple TCP connections or multicast UDP connections need to be established. Alternatively, the data might be sent to a single synchronization server which acts as a proxy for a number of other synchronization servers, thus synchronization server is then responsible for updating each of the other devices. In this thesis we mainly discuss one synchronization server model (the details will be presented in Chapter four). Finally after the changes are received by synchronization server, the next question is how the server performs the synchronization. These four questions will be used to structure discussion and design of a solution. Figure 2 shows an overview of the structure of data synchronization solutions.

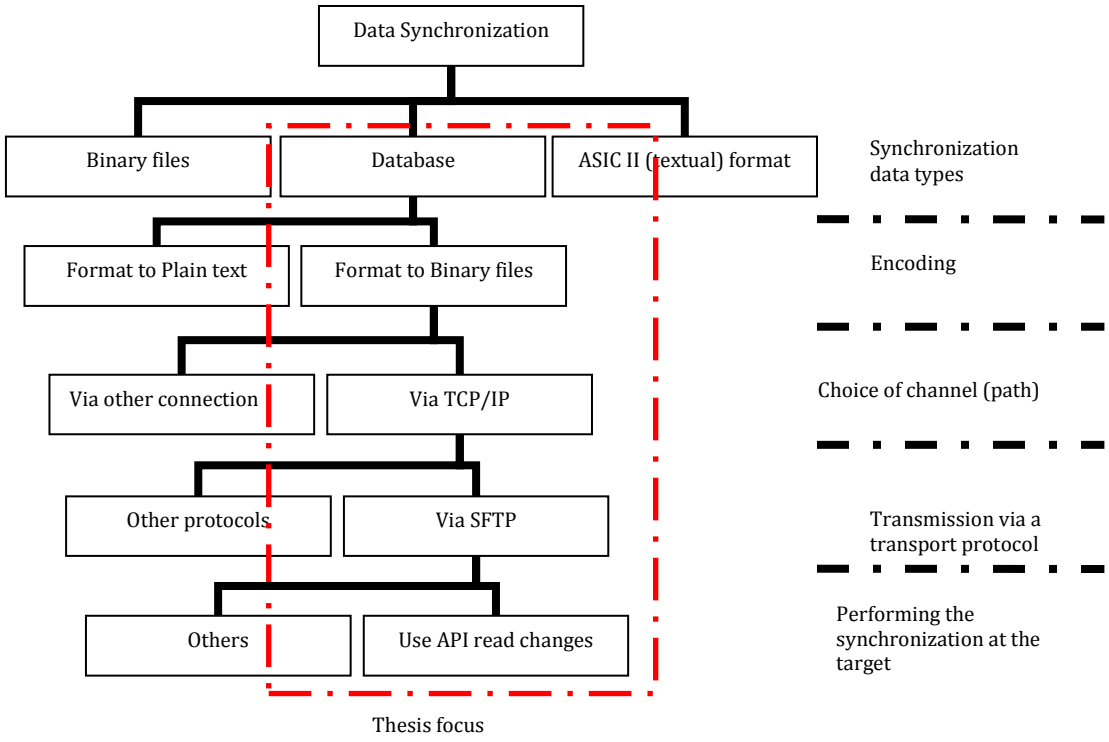

Figure 2 Overview of solution based on the questions above

The top layer of Figure 2 shows that an initial question is the data types that will be synchronized. In this thesis we divide these types in to the followings: binary files, database records and text files. The second layer shows two alternative ways of encoding the changes for transmission between the source device(s) and the target device(s). The third layer shows that there may be different paths for transmitting data between the source and the destination(s). In this thesis we will consider the case of using TCP/IP protocol. The forth layer indicates that different transport protocols can be used, like SFTP or HTTPS; (HTTPS represents a combination of HTTP with SSL/TLS over TCP; while SFTP represents a combination of a file transfer protocol with SSL/TLS over TCP). Lastly, the end device can have different ways to process the received data, such as use of a specific APIs to perform the actual data synchronization.

This thesis will describe mechanisms of data synchronization between Personal Computers (PCs) and consumer electronics, consumer electronics including smart phones or PDAs. The synchronization process between devices can be categorized into two groups. Firstly from a temporal perspective, the process can be real-time or non-real time synchronization. The second group concerns whether there is a single source device synchronizing with one target device or a single source synchronizing with many target devices. This thesis will mainly focus on non-real-time synchronization and a single source synchronizing data with many target devices. The thesis will motivate the need for solutions for handling non-real-time data synchronization using an example use case. In this use case the thesis will analyze a number of different solutions with their advantages and disadvantages and synchronization solutions will forces on working with databases records. Next, the selection of solution methods will be validated using measurements of an implementation of a non-real-time database synchronization system with SQLite database and .NET framework.

This thesis is structured into a number of chapters: This first chapter introduced the problem and gave an overview of this thesis. Chapter two introduced the concept of synchronization, along with a discussion of differences between the real-time and non-real-time synchronization. In chapter three, the thesis described non-real-time synchronization in more details and listed solutions for performing non-real-time synchronization, and then analyzes advantages and disadvantages of different solution to implement a non-real-time synchronization system. In Chapter four the details of an implementation and its design based on the .NET framework, SQLite databases and TCP/IP connections are presented. The performance of this prototype will be examined based upon analysis of the results of a number of tests. Chapter five presented the results of the previous chapters and draws conclusions. Additionally, chapter five presented some ideas for improvement to this prototype and suggestions for future works.

2 The concept of data synchronization

In this chapter, we will introduce the concept of data synchronization; including theories, single source and target synchronization versus single source and multiple targets, some widely used solutions for real-time and non-real-time synchronization; commonly used synchronization protocols for PIM database records; and the performance of different protocols. This chapter is mainly based on the paper by Agarwal, Starobinski and Tranchtenberg [5], Subversion Documentation [6], Concurrent Versions System (CVS) [7].

2.1 Fundamentals of Synchronization

Data synchronization is a process to maintain data consistency among source(s) and target(s). The data can be different types or formats, for example a database record, binary file, or textual file [5]. We will separate this process into five topics in the discussion below.

Firstly, the number of source and target devices: there might be only one source device and one target device to synchronize; This single source to single target paradigm is widely used for synchronizing data between a PDA and an especially PC for PIM data. Another example of single source and target paradigm is to synchronize media files between a media player and a PC, so that users can create their playlist using desktop or laptop and utilize the playlist and media files in their media player. In contrast, there can be more than two target devices which a single source should synchronize with. For example, users might have a media player used for jogging and another that used when commuting to and from work. There is also the possibility for more than two source devices to synchronize with a single synchronization server. For example, this paradigm is commonly used for file synchronization via a version control system, and sometimes it is also used to synchronize a PDA with multiple desktop PCs. Lastly, there can be many target devices synchronizing with many source devices; this model is more complicated than the other cases, and more related to The Peer-to–Peer[3] model. The example of this scenario may involve many PCs synchronizing media files with each other without a central media server. Figure 3 shows these different synchronization paradigms. Note that the clients can be acting as either a source or target or as both a target and source. For example in the case of source code version control system, the server is the target for clients which act as sources to provide their individual updates, then the server acts as a source to provide the updated files to any client which acts as a target. Figure 3 shows the three main synchronization paradigms: one device to one device, one device to more devices and more devices to more devices synchronization.

[3] Peer-to-Peer is any distributed network architecture composed of participants that communicate with each other to share resources (for example disk storage,) directly between the participants without the need for central coordination. [42]

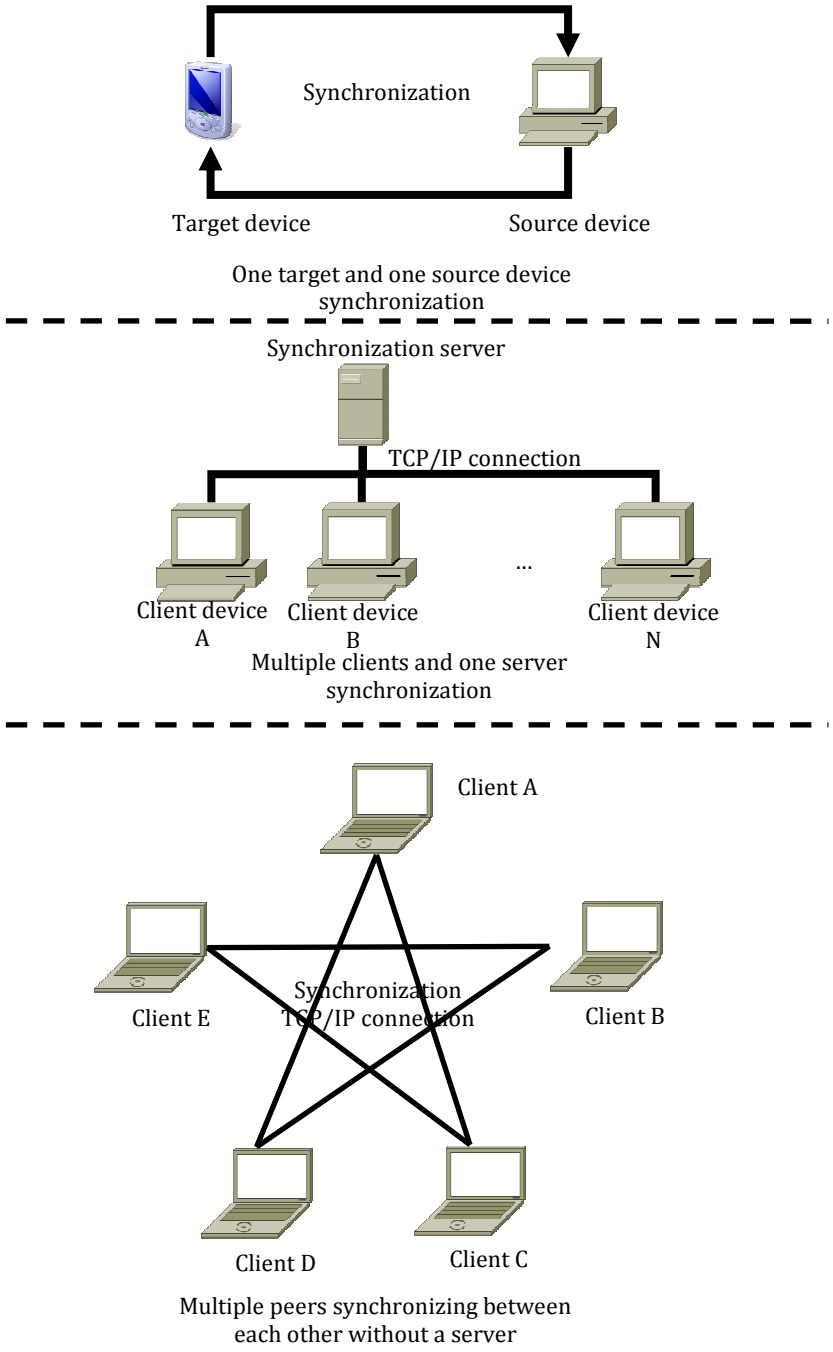

Figure 3: Different synchronization paradigms

The number of devices to be synchronized can lead to different synchronization approaches. One approach does not have a clear synchronization server and client structure; as both client and server can modify, delete, or insert the data in a local database or file system. This approach is commonly used in synchronizing PIM data between desktop PCs and hand held devices. The other approach has a clearly identified synchronization server and client structure. All of the clients connect to the synchronization server, the synchronized data is not accessed directly on the synchronization server, rather than a client synchronizes with the server in order to access the data. This structure is commonly used in version control systems.

As noted earlier different data types may be involved in the synchronization. As shown in the previous example, these are frequently binary files, textual based files and database records. In the case of the type of database records, the size of these database records is very small. As a result an application might support real-time synchronization by sending messages between devices when any change is made. In the case of a version control system the program's source code files are always in a text format. Because the changes to these files generally only concern few lines of the file, it makes sense to send only the additions and deletions in terms of lines between two versions of the files. This reduces the amount of information that needs to be transferred between the two machines and decreases the time it takes to synchronize. Lastly, there are binary files, such as media files, executable application files, images, video clips, etc. Binary files are generally synchronized as a single binary file (although the underlying system might split the file into chunks – for examples as is done in BitTorrent [8]). There are a lot of applications that perform binary files synchronization, some are media management applications, such as Apple's iTunes®[4] , DropBox.

When it comes to synchronization protocols: different synchronization protocols can be used in different situations. The most commonly used protocol for handing PIM data synchronization is Synchronization Markup Language (SyncML). SyncML is an open standard for synchronizing PIM contacts, task lists, and scheduling records between handheld devices and PCs or servers. A typical example is synchronizing a Windows Mobile based PDA with a PC running Microsoft's Windows XP OS using the Microsoft ActiveSync. [9] The SyncML protocol is frequently used over an USB connection, but it also works a suitably equipped PC and PDA with serial connection such as Bluetooth. SyncML can also be used over HTTP (which in turn uses TCP as its transport protocol), for instance, Microsoft's Exchange Server Protocols. The Exchange Server Protocols are exchange information between two applications running on computers connected via a network to accomplish predefined tasks. [10] This protocol frequently used by Windows Mobile devices iOS devices to synchronize PIM data or e-mail with Microsoft Exchange Server.

Another important factor in synchronization is the scalability of the synchronization process. The most common evaluation of scalability is the performance of a synchronization protocol or mechanism. This is a function of the time complexity of the protocol (or mechanism), where a short synchronization time reflects better performance. The performance can also be evaluated in terms of the following parameters: [5] data transmission load; end device's computational load; number of devices involved in communicating; robustness (with respect to errors in transmission); and the memory usage on the end device(s). We will examine each of these details below.

- The data transmission load is the amount of data exchanged between source (devices) and target (devices). For a given throughput over a connection, the data transmission load affects the time consumed for each synchronization process. The less data to be transmitted the better the synchronization protocols

[4] iTunes® is a free application for Apple's Mac and PCs. This application synchronizes media content between computer and the user's iPod, iPhone, or Apple TV.

are. The pattern of communication may also be important – as the duration of the connection might affect both cost and battery power consumption. This is particularly important for handheld devices.

- For end device the computation load refers to the processing load on the end device during the synchronization process. A lower computational load not necessarily indicates a better protocol, since a higher computational load might be acceptable to reduce the amount of data to be transferred, especially if the cost of transferring data is high. Additionally, because different devices have different performance processors it may be desirable to shift the processing load to specific devices - if a device has a high performance processors, it will reduce the load on the other device involved in the synchronization. However, if the processing load on a handheld device is too high, then the protocol may not be acceptable because of the limited battery power available to such devices.
- The number of devices that communicate together at one time (which might be called the synchronization network size) refers to the ability of the synchronization protocol to handle a number of nodes. For example, Palm's HotSync protocol only supports one PIM database synchronizing with a single PDA, thus if a user has two or more PDAs they cannot use this protocol to synchronize their PIM information across these devices. In contrast, Microsoft Exchange protocol supports more than one device being synchronized with one or more source severs.
- Robustness refers to the error handling ability of the synchronization mechanism. If the mechanism has a single point of failure or is unable to handle message errors during the synchronization process, then the synchronization protocol is not robust. Synchronization protocols that transmit text based messages always have higher robustness than binary messages, as there is higher redundancy in the text message. However, to achieve high robustness we need to utilize some sort of error detection recovery or correction mechanisms. This error detection and correction could be provided by the synchronization protocol itself or by a lower layer protocol.
- The amount of memory needed by end device during synchronization is important because handheld devices do not all have large amounts of memory. Therefore this parameter is particularly important for handheld devices. As the amount of memory increase this parameter becomes less important.

This thesis will study synchronization process in terms of the parameters above; synchronization models, file types; standard synchronization protocols, and exchanging messages between two devices.

2.2 One to One synchronization model and solutions

2.2.1 One source to one target device synchronization model

In the "one to one" synchronization model two devices synchronize data between themselves. This model is mostly used for synchronizing PIM data between a PDA and a PC. Here we will consider an example of this simple data synchronization model - one source device and one target device in a synchronization scenario. In this scenario the target device is always synchronized with the same source device. Furthermore we

assume initial the source device is a desktop PC, the initial target device is a PDA, and the data to be synchronized between PC and PDA are PIM database records. Figure 4 shows an update of a PIM contact. The first portion of the example shows a PIM contact (for Mr. Long) being added at the PC and this is propagated to the PDA. In the next portion of the example, the user deletes an entry (for Mr. Green) form the PIM contact database on the PDA. And the next step, it synchronizes with the PC, at this stage, this entry is deleted from the PC's copy of the PIM contact database.

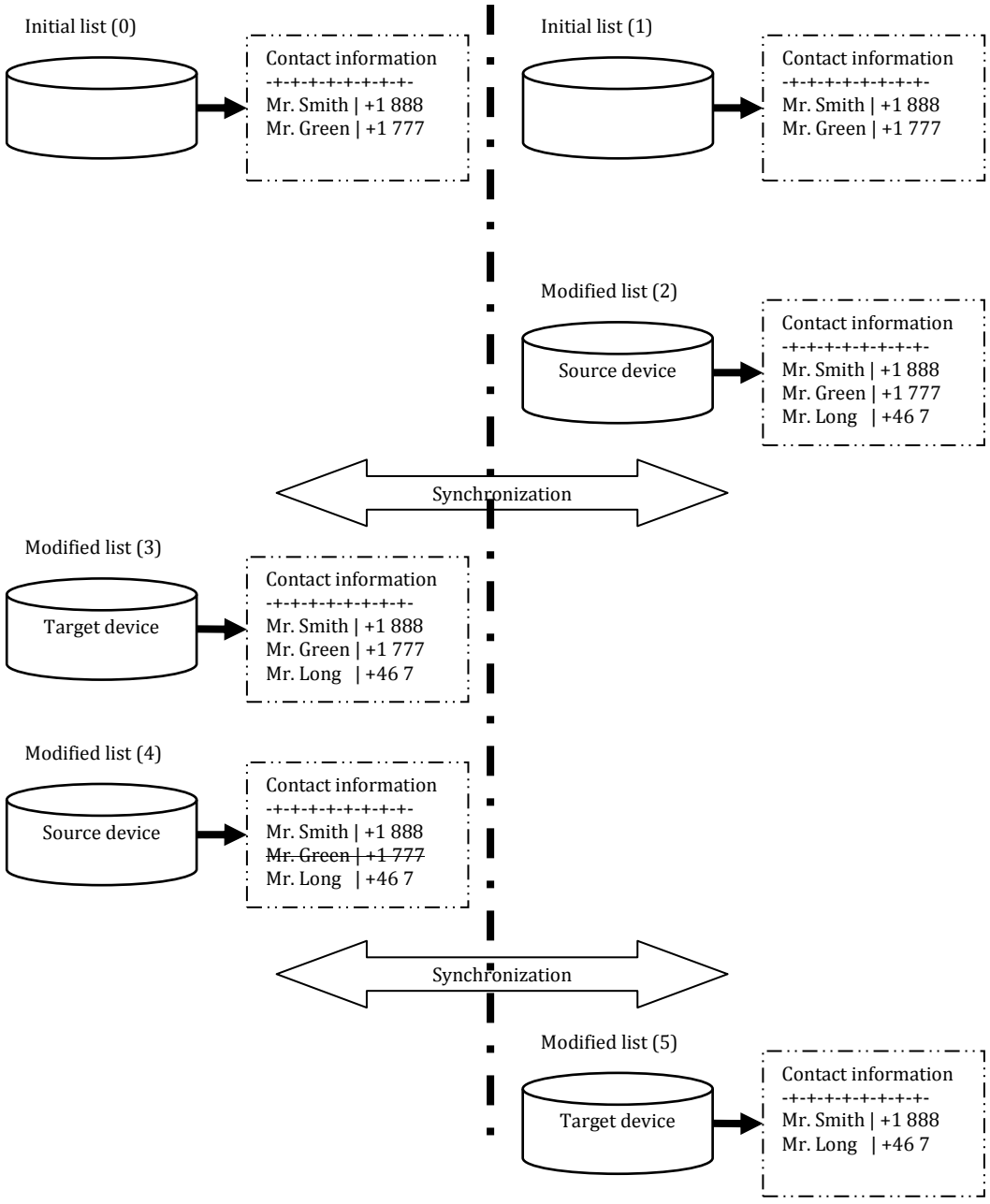

Figure 4: PIM contact information is modified and synchronized between the source device and a target device

In the scenario shown in Figure 4, initially both source device and target device have the same two contacts records in their respective contact information list, there are shown as “Initial list (0)” and “Initial list (1)”. Next one contact record (for Mr. Long) is inserted into the source device. After inserting this synchronization of record occurs, the new record is inserted into the list of contacts in target devices. Following there is a record is deleted from the PDA, this now becomes the source device, thus in the next synchronization the record will also be deleted from the target device.

2.2.2 Synchronization solutions

To implement one-to–one synchronization, it can be implemented by maintaining synchronization flags or timestamps. The synchronization time-stamp solution is commonly used to mark the records changed in the database. In order to store this time-stamp, an additional field is implemented in each record of database to store the time of its update. Normally the data type for this new field is the “time stamp” format. Figure 5 shows an example of using the time–stamp solution for synchronization. This example uses the same data as in the scenario above.

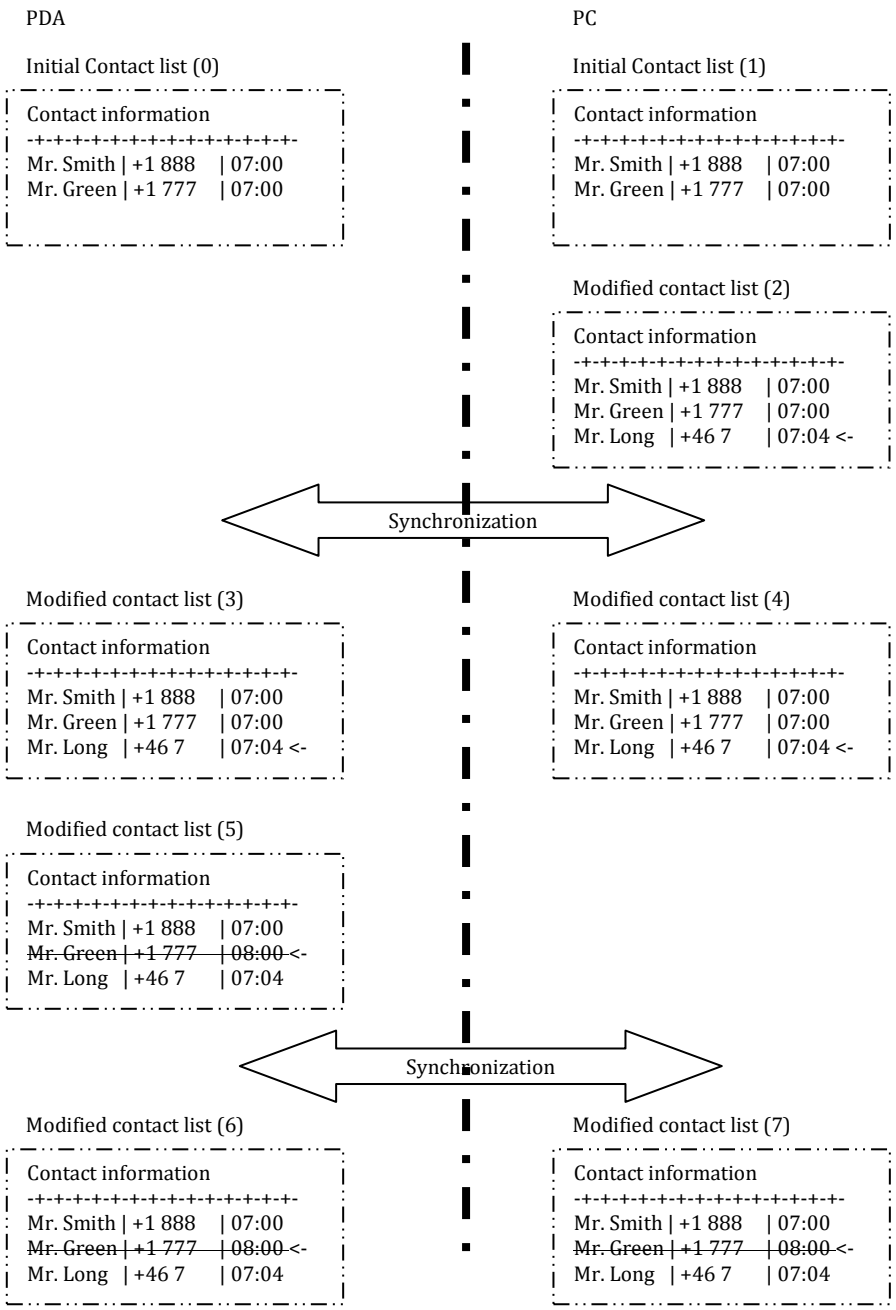

Figure 5: Time-stamps for one source and one target device synchronization

In the example shown in the Figure 5, there is additional field, which contains the time of the last change in any PIM record. In the initial case, there are two records in the contact information table, which both have the time-stamp of "07:00". Next a record is added to PC at 07:04. After this the synchronization occurs. During the synchronization process, both devices send each other their latest time-stamp, and check if these latest time-stamps are equal or not. If the latest time-stamps from both PC and PDA are equal, this means that there are no records need to be synchronized. Otherwise, the device with later time-stamp become the source would send all of its records that have been changed since the time of the last synchronization to the other device – excluding those that have just been added due to the first phase of the synchronization. Now both

devices update their time of last modification and synchronization to the current time. In the Figure 5, after synchronization both devices have same records.

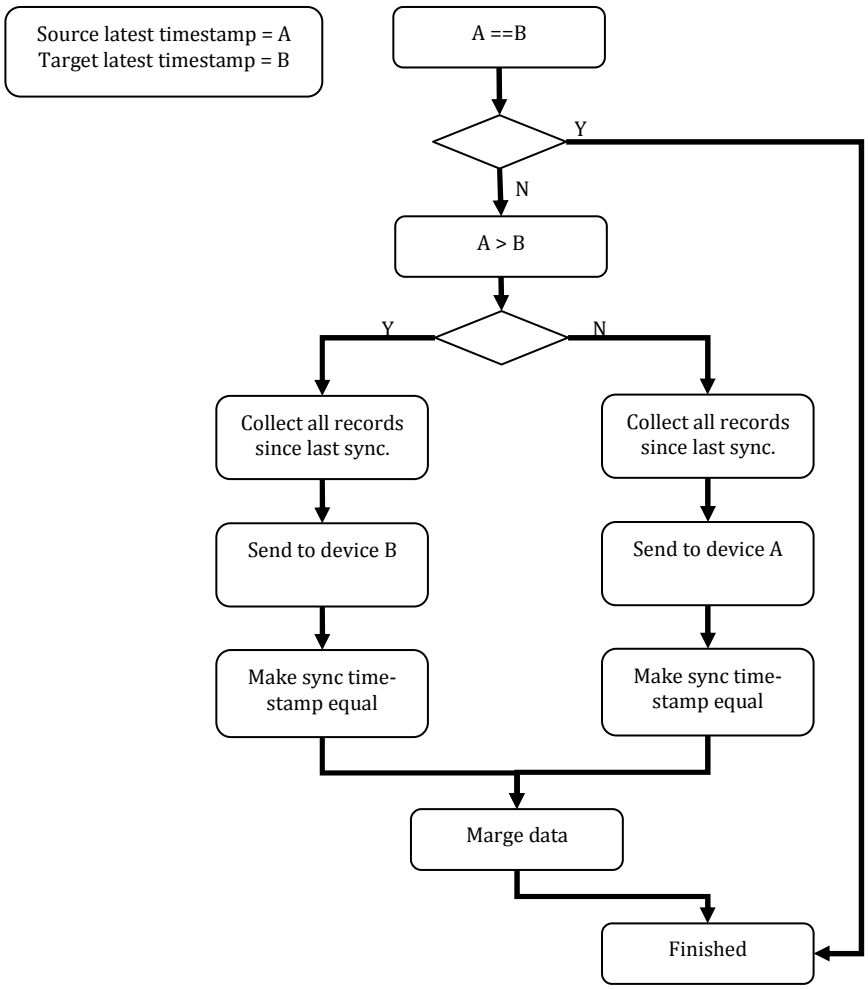

Figure 6: Synchronization determination procedure

As shown in the Figure 6, the first step has both devices send messages in order to compare their latest time stamps, if the latest time stamps is equal, the synchronization process finished, as no changes have been made on either device. If the latest time stamps are not equal, the synchronization mechanism has to merge the changed data from both devices. An alternative method other than using time-stamps to maintain synchronization process is to use synchronization flags.

Figure 7 shows how synchronization flags works. It utilizes the same scenario as used in the previous example. As shown in the Figure 7 an additional field has been added to each record in the database. This field contains a flags which indicates if this record has been synchronized with the other device or not. In our case, Zero (0) indicates that it has not been synchronized, while a value of one (1) indicates that the record has been synchronized. When a new record is added to the database, the flag is set to zero indicating that the new record has not yet been synchronized with the other database. After synchronization the flag will be set to one.

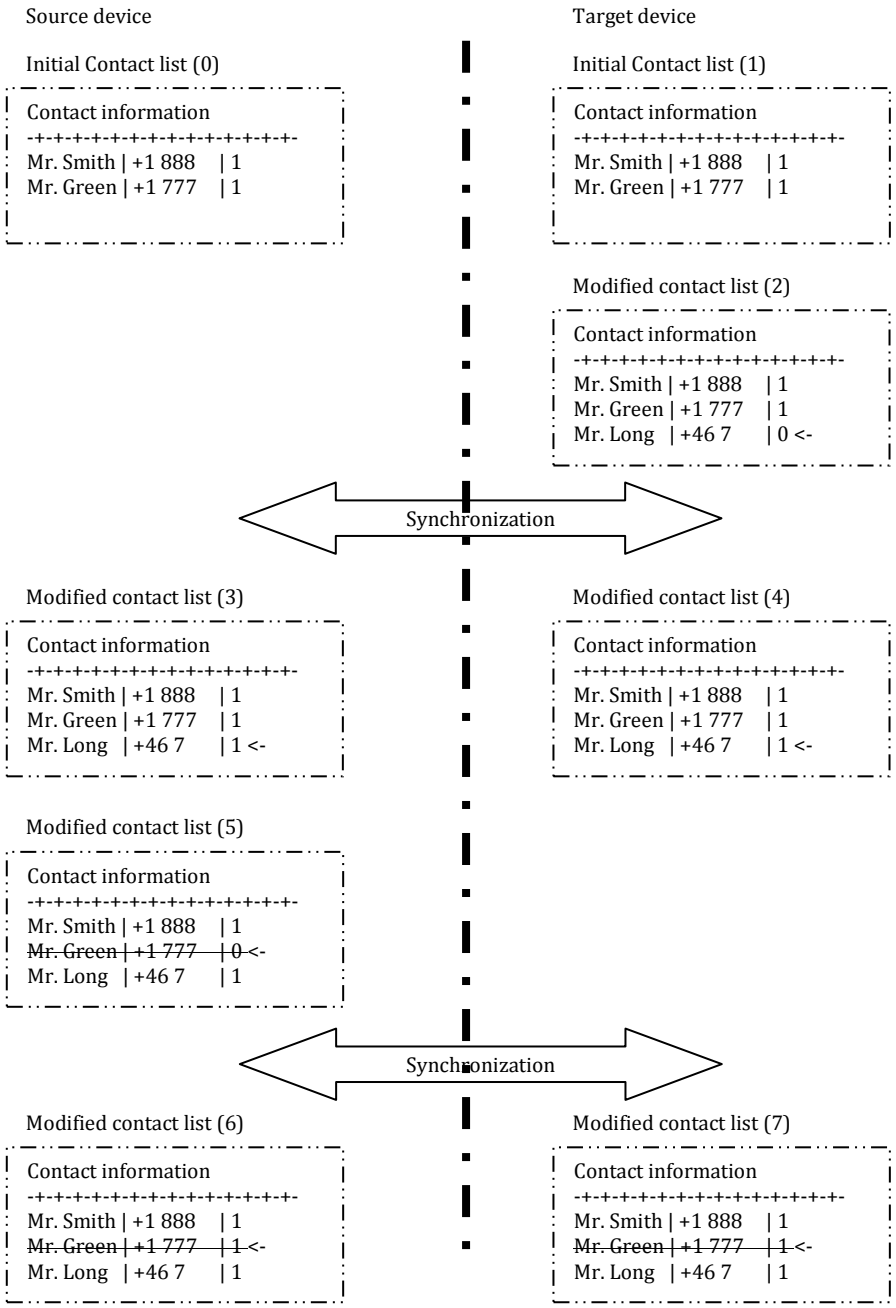

Figure 7: Using synchronization flags for one source and one target device synchronization

The Figure 8 shows the working flow of synchronization flags solution.

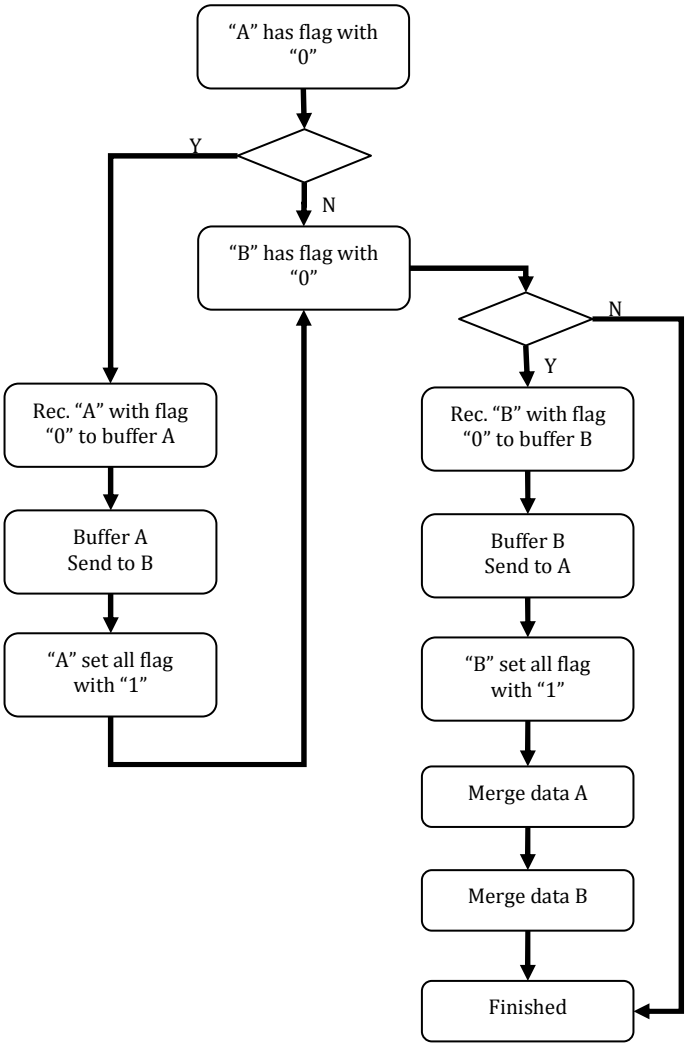

Figure 8: Synchronization flags solution working flow

In the Figure 8, this mechanism only checks the synchronization flags, then, moves all the records with synchronization flags set to "0" to a buffer, send the buffer to other device, and the target device merges these records with its own data. This approach requires additional bit of storage per record in order to store the synchronization mark data. Compared with the time-stamp solution, which requires four bytes of extras storage per record and two times four bytes for the time stamps of the last synchronizations process, [11] the solution using synchronization flags requires less memory space. Therefore marking record in the database in order to keep data consistency between source and target device, using synchronization flag has less data consumption than using a time stamp. However, the use of synchronization flags is only applicable to the case of two devices and does not extend to do synchronization of multiple devices. Additionally, one should consider the number of records that are likely to be involved in a PIM –in most cases- this number will not be a relatively big number, hence storage consumption between synchronization time-stamp and flags is be considered as a major issue.

The example above mainly focused on one source device synchronizing with one target device. However, if more than two devices are involved in the synchronization process, the case becomes more complicated.

2.3 Fast and slow synchronization

This section introduces two synchronization models called “fast synchronization” and “slow synchronization”.

2.3.1 Fast synchronization model

Fast synchronization is most often used when a target only synchronizes with a single source device, however it also can be used when one device synchronizes with more than one device. In this case the target device uses synchronization flags or time-stamps to mark records that have been modified since last synchronization, the synchronization flags or time-stamps are saved with each record. When synchronization starts, the synchronization flags or time-stamps will be checked. Source devices will compare these records with the records on the target devices. After that, one of the following operations will occur: The target device’s modified record(s) will be inserted to the source device; the target device’s record(s) will be replaced by record(s) from the source device; or the appropriate record(s) will be deleted from the source or target as appropriate. After synchronization all the synchronization flags will be reset (i.e. set to the value one). Note that, during the synchronization process, the computation work, such as comparing the synchronization flags or time-stamps, should be done on the more powerful CPU machine. For instance, if the synchronization is done between PC and PDA, most of the computational work will be done on the PC side.

However, if there are more than one target devices or source devices in the synchronization process, as shown in the Figure 9, the synchronization flags in used the fast synchronization model cannot be used. This is because the synchronization flags will be reset when the target device is synchronized with first source device; hence there are no synchronization flags to be used for target device synchronizing at the second source device connected. Although the synchronization flag concept could be extended to have additional flags for use with multiple devices, the use of time stamps seems more appropriate for use with multiple devices.

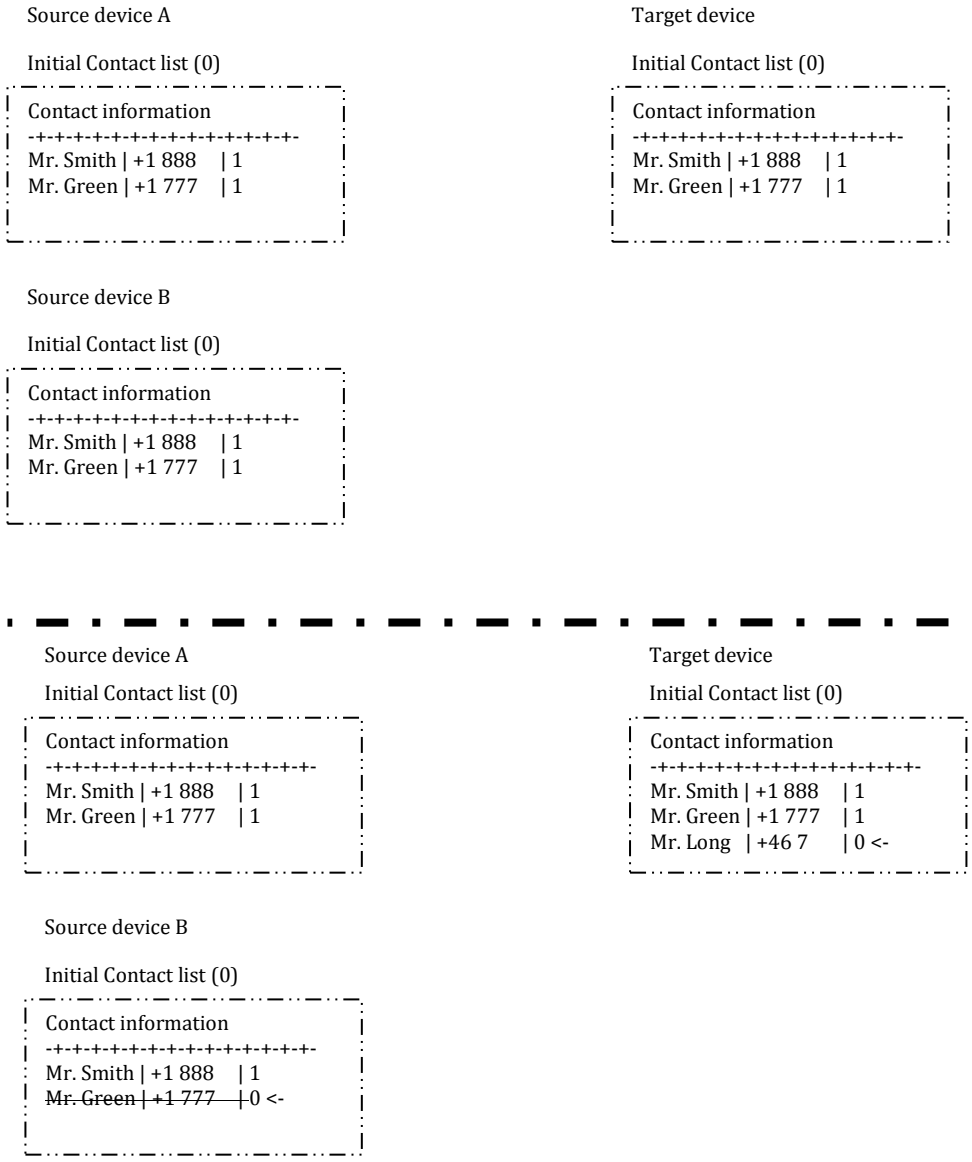

Figure 9: Contact information changed on both source device and target device at the same time

In the Figure 9, there are two source devices. If the changes have been made on both of the source devices and target device, then synchronization flags will not work. To address this problem we will introduce the slow synchronization model.

2.3.2 Slow synchronization model

For the cases that the fast synchronization model cannot handle, slow synchronization should be used. More specifically, as long as there is more than one source device or target device and more than one change made at a time, slow synchronization should be used. During the synchronization process, the whole database will be sent from one device to another in order to compute the changes. In the most of the case, the device with more power CPU and memory capacity will take responsible for more computation work.

The Figure 10 shows how slow synchronization is processed.

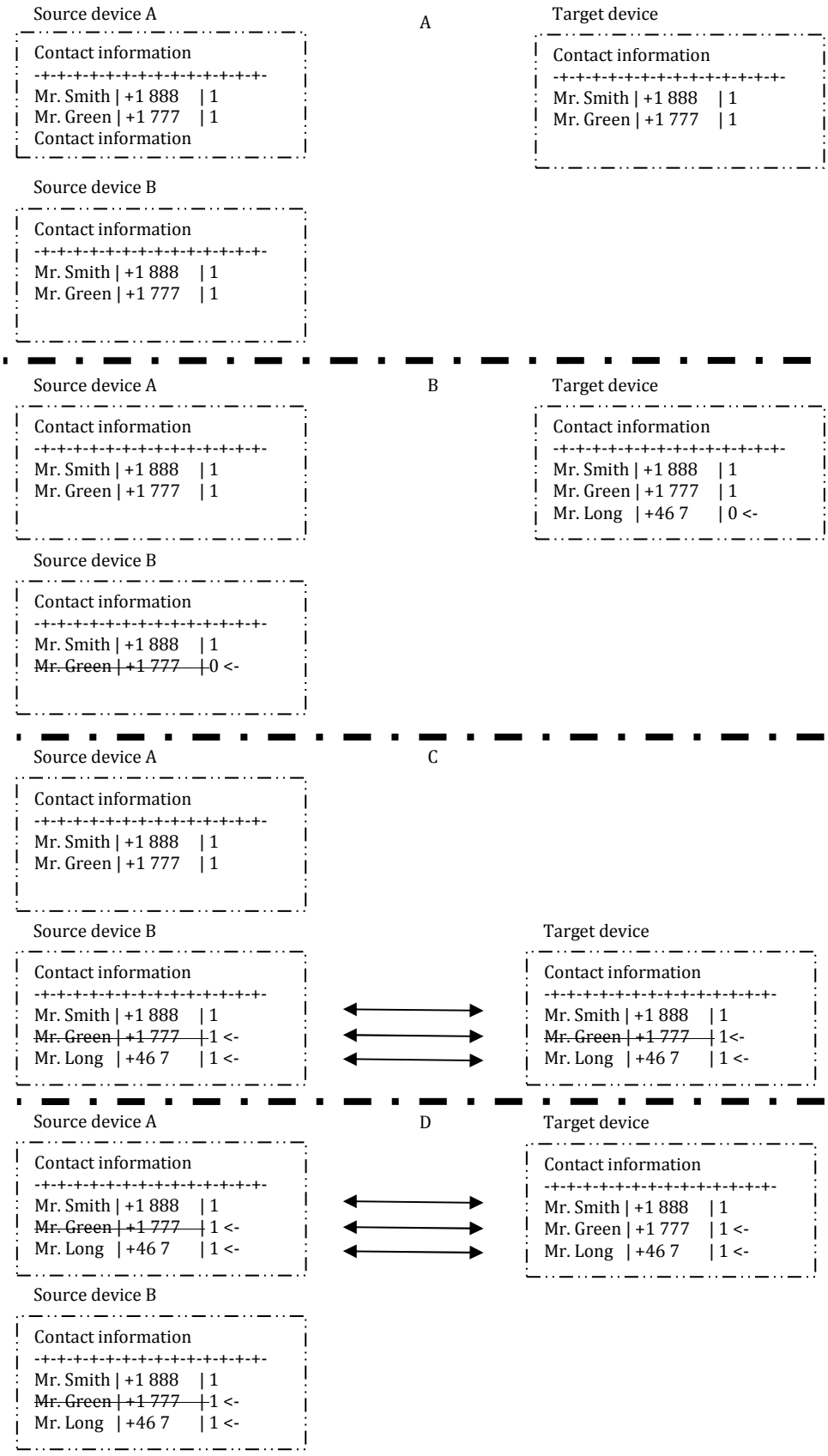

Figure 10: Slow synchronization

Figure 10 showed how changes on either the source device or target device are propagated using slow synchronization in the sections C and D. During slow synchronization, all the records are sent to one of the devices in order to find all of the differences since their last synchronization. This means that slow synchronization takes time. It has a higher latency and higher bandwidth usage than fast synchronization. Since slow synchronization processes all of the database records during the synchronization process, the time complexities of slow synchronization increase linearly with the number of records in the database, while, fast synchronization only needed to transfer the data that was actually changed, so the bandwidth required only depends on the number of modified records.

Both fast and slow synchronization are mainly used for database record synchronization, and during the synchronization process, the messages sent between source device and target devices are generally encoded as text records.

2.4 Database transaction log based synchronization

In a database that keeps transaction logs, it is possible to simply reply a log from a given time point (the time of last synchronization) in order to synchronize the two databases. This is because the transaction log contains all of the changes that have been made. This process is linear in the number of changes and does not require accessing all of the records in the database. However, it requires additional storage for the transaction log.

2.5 Repository based synchronization system

In the previous sections we have looked at the case of synchronizing a database (specifically PIM data) between devices. In this type of synchronization the changes could be made on any of the devices and the database in all devices should be brought a consistent state. In this section we will consider another common case, repository synchronization, where there is a dedicated system that is to store (as a central repository) all of the information. In contrast with the previous synchronization models, the files will not be changed independently on the central repository. Users check out the data from the central repository in order to read and modify the data. This approach can be used for files, both text files and binary files. This approach to file synchronization keeps the files consistent - but only when they have been check-in and are checked back out. This model is commonly used for version control and file backup systems. There are many practical implementations of this approach to file synchronization. The solutions are used for data synchronization between PCs or between PCs and servers. Examples of this approach can be seen in the tools used for version control, such as CVS[5] and Subversion[6].

[5] The Concurrent Versions System (CVS) is an application revision control system widely used for distributed software development. It keeps track of all files changes, such as code insertions, deletions and modifying. It allows multiple developers to collaborate, while preserving the edits that have individually been made. [7]

[6] Subversion (command name "SVN") is a revision control system. Subversion can be used to maintain current and historical versions of files such as source code, web pages, and documentation. [46]

2.5.1 Version control system

A version control system can be defined as a centralized system for sharing information which exists in distinct versions. The core of a version control system is a repository. The repository is a data storage center. The repository stores information typically files in a directory hierarchy. In a version control system, the central repository can be seen as a source device and all the clients connecting to the repository can be seen as target devices. Any target can connect, read and write data in the repository. When writing data into the repository the clients send information to the repository, thus making this data available for all the clients. Figure 11 shows a typical structure of version control system

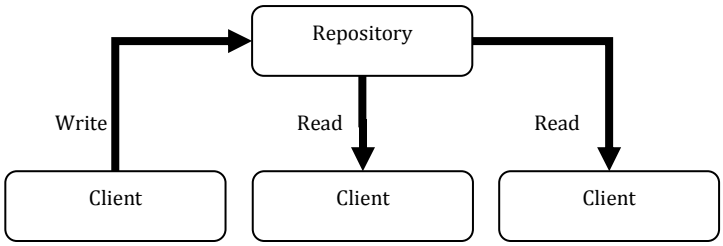

Figure 11: Typical structure of a version control system.

As Figure 11 shows, one client writes data to repository and the other clients can read this data from the repository. Although this hierarchy looks similar to a common file server, if offers more advanced functions than a typical file server, the repository in a version control system has mechanisms that enable the repository to record every change, such as inserts, modification, and deletions, to the files managed by the repository.

For instance, when a client reads a file from repository, by default, the client will receive the latest version of the file from the repository. In addition to checking out the latest version, the client can also view the previous states of the file. The client not only can see what changes were made to this file, but can also see who made these changes to this file.

A version control system provides clients with a file synchronization function. Additionally, the version control system also has to solve the problem of maintaining consistency while merging multiple changes to a file, thus, the system has to allow clients to share information, but prevents clients from accidentally stepping on each other's feet, i.e., preventing clients from overwrite each other's changes. [6]

In the database synchronization discussed in the previous section, records could be changed on both source device and target devices. In contrast, in a version control system, changes cannot be made directly on the source (repository) machine. The source device in this case updates the repository, and then end users at each of the clients have to check out the files from the repository. These client devices can make changes themselves to their local copy of the file, after that, each client will send their

changes to the repository. The version control system uses “a copy modify merge solution” and “lock-modify-unlock” in order to keep the data consistent between different clients. Figure 12 and Figure 13 show how the “copy - modify –merge” solution works.

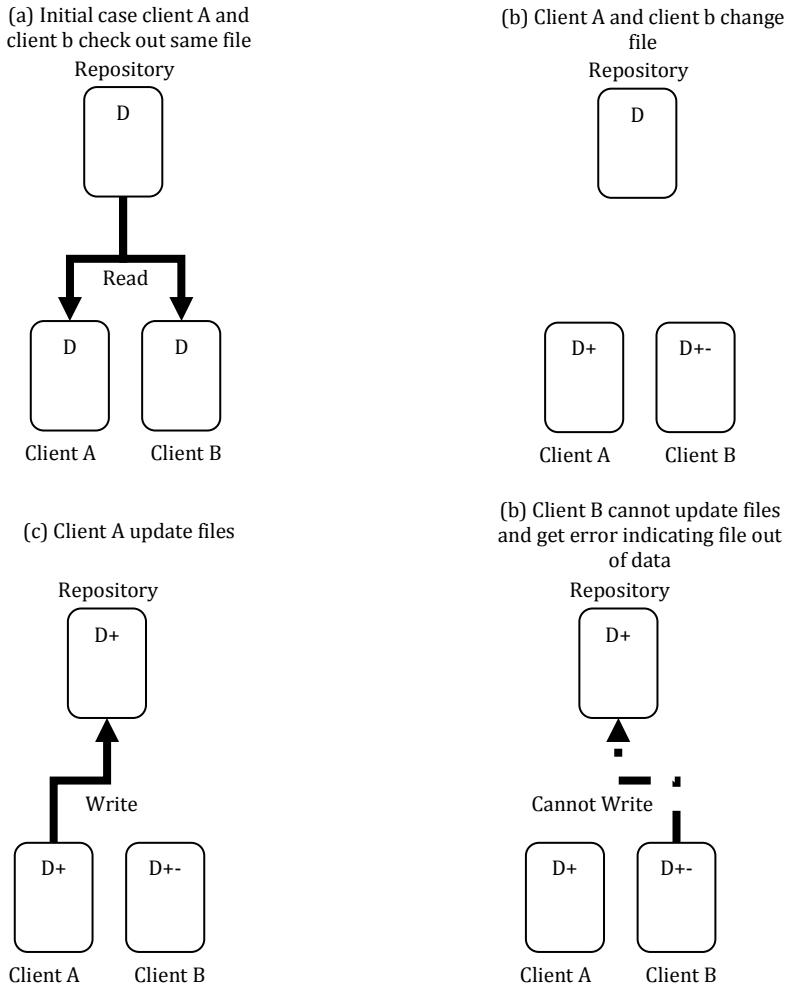

Figure 12: Copy - modify –merge solution work (where “+” indicates an insertion and “-” indicates deletions)

As Figure 12 shows, initially both client A and client B check out the documents “D” from the repository. After that, both clients change the document. After completing its changes client “A” sends its changes to the repository, as long as no other clients have changed this file, it becomes the latest version of the file “D”. When client “B" tries to change files “D”, the version control system will report a “file out of date” error to client “B”. Now, client B has to fetch the latest version of the files from the repository server compute the differences between the local version and the latest version of this file, and then send its changes to central repository. Figure 13 shows how client B fetches the latest version of the files and merges it. Finally client fetches the latest version of file D, now both clients have a consistent copy of the file.

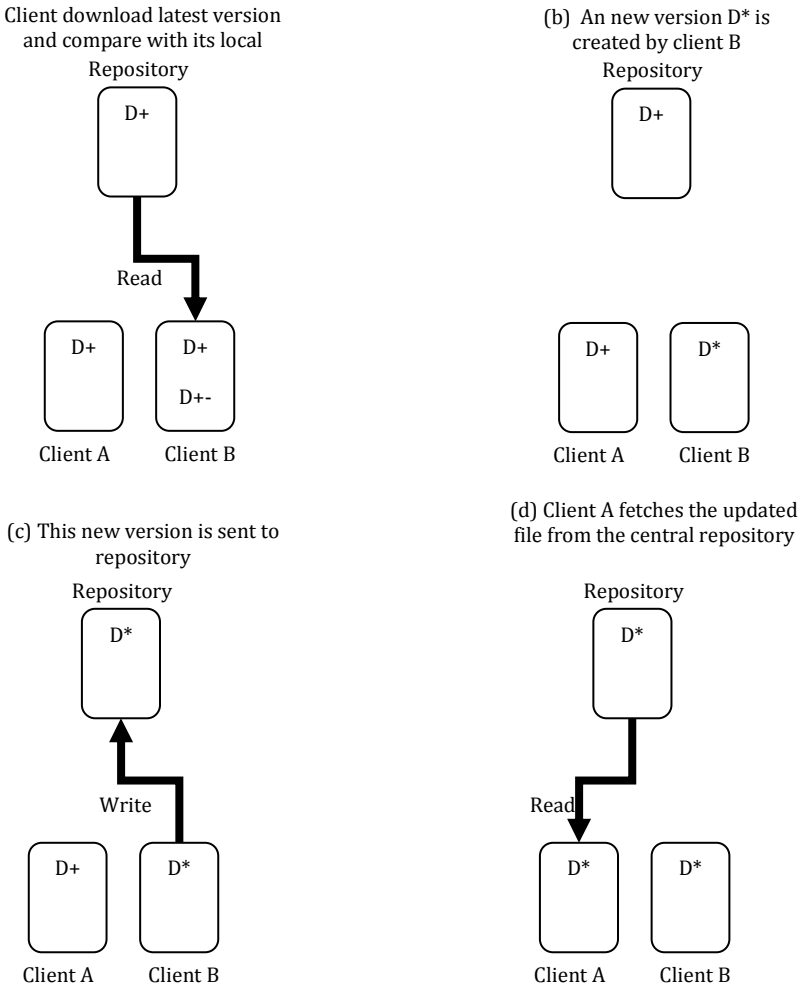

Figure 13: Copy - modify –merge solution works continued

Figure 13 begins with client B fetching the current version from the central repository in order to compare it with client B's local version of the files, after client B's local file and the file on the repository are compared, client B merges its changes (b). Next (c) client B sends it's a new version of the file to the repository replacing the previous version of the file. In the final step (d), the version control server will inform client A to fetch the latest version of the files in order to synchronize its copy with the repository's copy of the file.

The copy-modify-merge solution can work extremely smoothly in practice. The files in the repository can be accessed and edited in parallel without waiting for other clients. When clients work on the same files, the "copy – modify –merge" solution can maintain the concurrent changes provided that the changes do not overlap with each other. Although conflicts can happen, they can generally be solved without doing too much computational work. However, some conflict resolution may require the user to manually resolve the conflict.

This copy-modify-merge solution is always used for text file synchronization, since each word (or line) in a file is distinct. Therefore during the synchronization process, only differences between the files need to be sent between the clients and repository. Since the version control system utilized a TCP connection (for reliability) network it is

possible to provide synchronization between any computers which have TCP connectivity. This generally means that while the clients can be behind a network address translation (NAT) device or firewall, but the server cannot be unless it has some port forwarding mechanism. Additionally, this means that the clients have to remain connected to the repository server in order get notifications.

As we can see, a version control system is more flexible than the previous solutions that we examined while considering the synchronization of one source device and one target device. The copy-modify-merge solution needs more computational resources, but allows more devices to be involved in the synchronization process.

2.5.2 File synchronization (backup) system

In the previous section, we examined a version control system. In most of the time, version control systems work with text files. Text files are easy to edit typically with an editor application. However, binary files are quite different as generally, they cannot be easily modified by common applications. Therefore, rather than performing synchronization based on changes to the binary files we copy the entire binary file between the source device and target devices in most of the cases. There are few tools that compute differences and uses these differences for synchronization of binary files, for example rdiff-back [12] and duplicity [13]. However these applications are rarely used.

In order to maintain binary files consistency, a binary file synchronization system can use the Lock-Modify-Unlock approach. This approach is also used in some version control systems for text files, but it is more useful for binary file synchronization. In this case, files the synchronization system also needs a repository.

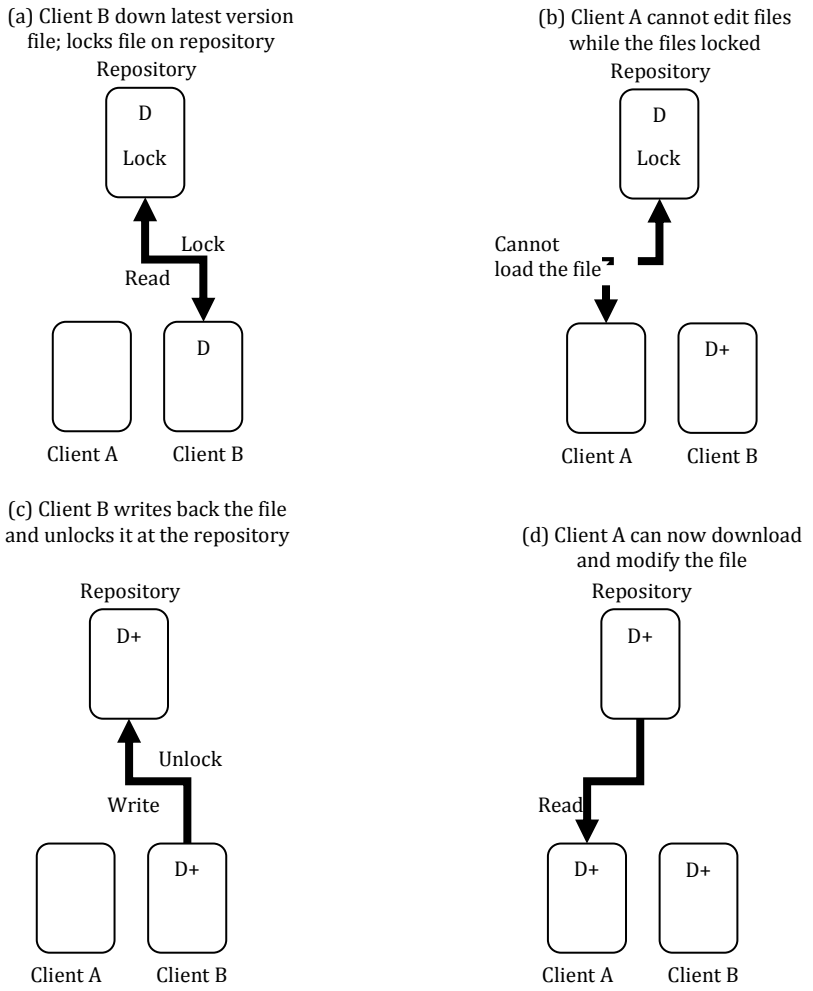

Figure 14: Lock-Modify-Unlock Solution

In the case of Lock-Modify-Unlock, as shown in Figure 14 (a) client B checks out a binary file from repository and locks the files at the repository in order to prevent other clients from modifying the checked out file. A shown in Figure 14 (b) client A cannot download the file from the repository. As show in Figure 14 (c), when the update of the file is complete, client B sends the new version of binary file to the repository and unlocks in the files in the repository. Now that the file is unlocked, other clients can check out the latest version of the binary file.

This solution is able to maintain binary file consistency. However, it cannot provide real-time synchronization and the synchronization process always takes time proportional to the amount of time that it takes to transfer the file and to perform the processing on it and to transfer the resulting file back to the repository. The Lock - Modify - Unlock solution always utilizes a reliable connection (such as TCP or SCTP). File synchronization is also commonly used for home backups to external hard drives or USB flash drive(s); however, in these cases the connection is over an USB but not a network.

2.6 Synchronization protocols

A synchronization protocol must define a message format and procedure for synchronization process. For example, the target device(s) may not always be connected

to the source device(s). When users want to synchronize the devices they must connect the source device and target device via a network connection or cable. This connection can occur explicitly when the user wants to explicitly perform synchronization or the process could happen periodically or according to a pre-determined schedule. During the synchronization process, the target device learns about the changes made on the source device(s) (and vice versa). While the devices were disconnected, occasionally, both source device(s) and target device(s) need to resolve conflicts among the changes. This reconciliation process through which conflicts are resolved is part of the data synchronization process. [14] The result of the synchronization process is to make the source and target devices of data look identical.

A data synchronization protocol defines the communication workflow during the data synchronization session between the target and source devices. The synchronization protocol has to identify the records that have been changed and resolve the records that have conflict issues. Note that for data that does not have explicit record structure the protocol must determine what size objects will be examined, compared, and synchronized.

During the synchronization process, the bandwidth between the source devices and target devices is limited (even in the case when the source and target are interconnect by a high speed network or bus[7] there is some limit to the throughput between the devices). In order to minimize the required bandwidth, both the source devices and target devices would encode and compress their messages. If the devices are connected by a connection that has a high cost per packet, the protocol should minimize the number of packets sent between the source and target devices. If there is high network latency, then the protocol should attempt to utilize sufficiently large buffer and flow control that it can effectively utilize the available bandwidth without causing congestion. Furthermore, connection reliability may be an issue, if a disconnection occurs the protocol should respond appropriately. Depending upon the probability of disconnection the protocol may even need to resume the transfer of a file from the point where the file was successfully transferred to avoid unnecessary packet charge, and avoid unnecessary delay in completing the synchronization.

When defining a synchronization protocol, there are three important points to be defined, firstly the synchronization architecture, next the communication protocol, and finally the message format. In the following subsection we will examine, SyncML as an example of a typical synchronization protocol.

2.6.1 SyncML protocol

SyncML is an open industry initiative supported by many companies including Ericsson, IBM, Lotus, Matsushita, Motorola, Nokia, Openwave, and Starfish. The purpose of SyncML is to provide a standard for data synchronization across different platforms and devices. [15] It is a widely used synchronization protocol between PDAs and PCs, specifically for exchanging PIM data. A later version of the specification supports Push

[7] The bus is a subsystem that transfers data between components inside a computer, or between computers.

email. The messages sent between source and target device contain only modified information. Figure 15 shows the framework of SyncML protocol.

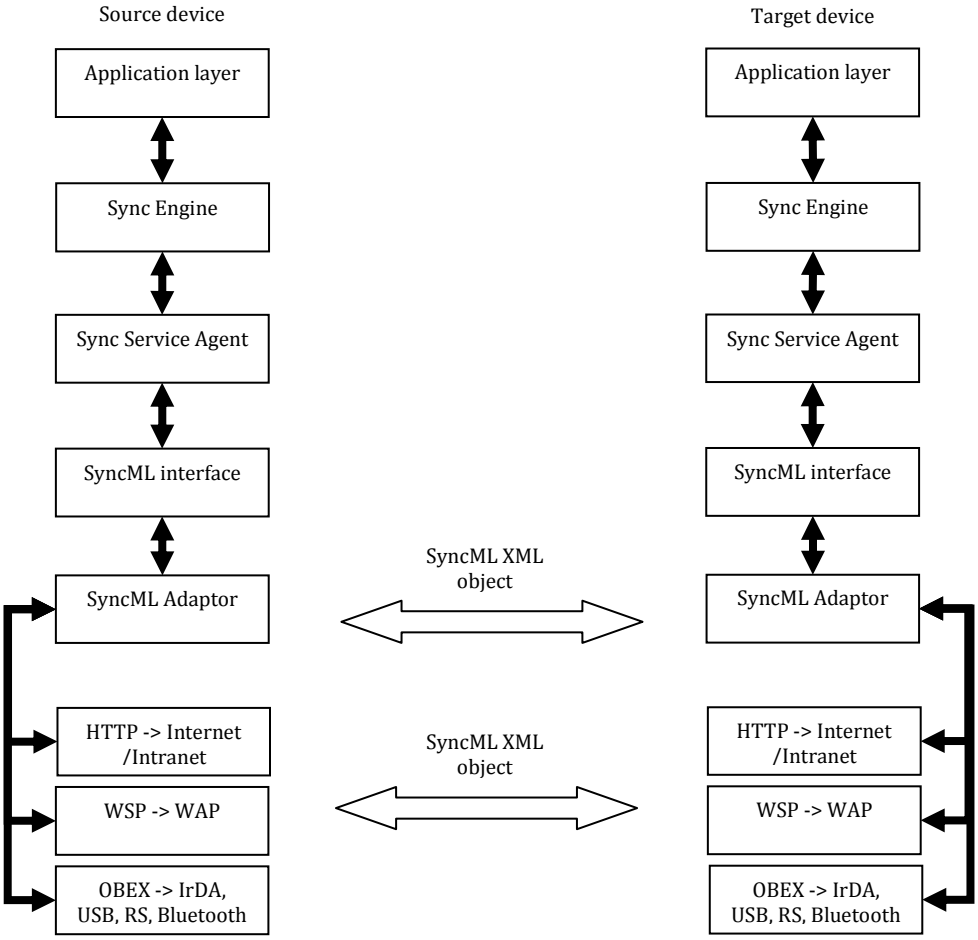

Figure 15: SyncML protocol Framework

Figure 15 shows how a synchronization service is provided by SyncML protocol. The data exchange between two devices can be done directly via the SyncML adaptor connection, via TCP/IP or a short range wireless connection (such as via Bluetooth). In this framework, the 'Sync Engine' functionality is placed in both target and source devices; this means both target and source devices must have the ability to do the necessary computational work, specifically to compute the differences between the data at the devices. Since SyncML is frequently used for PIM data placed on PDAs when synchronizing with PCs. In this case PDAs often have limited processing power; thus the sync engine is always only executed on the source devices. The 'Sync service Agent' uses the protocol defined in the representation protocol [16] which is offered by the SyncML interface to talk between source and target device. [17]

Since the SyncML protocol was original designed for synchronizing PIM data between PDAs and PCs, there are many common data formats related to PIM data that are pre-defined by SyncML, such as vCard for contact information, vCalendar and iCalendar for calendar information, “To Do Lists” and “Journal” information. In addition to PIM

information, SyncML now has definitions for e-mail and network news. All of the messages exchanged between source and target device(s) are based on the eXtensible Markup Language (XML) according to the SyncML standard Representation Protocol Document Type Definition (DTD)[8].

Although SyncML has many well defined data formats, most of them are related to PIM information. If the data to be synchronized is not PIM information, then a better solution would be to exchange DTDs for the new objects which they are to be exchanged or to utilize another synchronization protocol.

2.7 Real-time and non-real time synchronization

Synchronization can be divided into two categories: real-time synchronization and non-real-time synchronization. Real-time synchronization always requires a continuous data connection between source and target devices. This type of synchronization offers the best to maintain data consistency between source and target devices, since any changes to the data in one device will be rapidly propagated to the other device(s). In contrast, non-real-time synchronization performs the synchronization at a specific time. Version control systems and file synchronization systems are non-real-time synchronization systems. The systems considered in chapter 2.1, with one target device and one source device can utilize either real–time or non-real-time synchronization depending on the system.

Real-time synchronization monitors the records in the database or the files in directories for changes and replicates the changes soon after they happen. Since the sources device storage areas are always monitored, there is no need for a special mechanism to build a synchronization table, however, such tools often provide a way to indicate with entities are to be synchronized or excluded. Real-time synchronization can provide data consistency in an efficient way; however, real-time synchronization may not be appropriate in all situations. [18] It needs continuous connectivity between the source and target devices, hence implementation of real-time synchronization may be based upon use of remote procedure calls (RPCs)[9], and thus it means that providing reliability is necessary for this utilize.

2.7.1 Continuous Connectivity

Continuous connectivity can be provided by using a cellular network data connection or via a fixed network connection. 3G cellular networks are widely used all over the world and today such networks can provide a stable data connection over a wide area with a maximum data rate of up to 7.2 Mbps. Furthermore, the Swedish telephone operator "TeliaSonera" has already launched a 4G mobile broadband service in some cities within the Nordic countries. These 4G cellular networks offer throughputs of up to a maximum of 100 Mbits. [19] While such mobile broadband connectivity may fulfill the requirement for connection throughputs, other factors have lead to them not being used. Until recently, these mobile networks have not been particularly successful mainly because of the retail service prices and the unreliability of receiving the *expected* data rate.

[8] Document Type Definition (DTD) is a set of markup declarations that define a document type for SGML-family markup languages (SGML, XML, and HTML). A DTD is a kind of XML schema. [44]
[9] A remote procedure call (RPC) allows programs to call functions, which execute in another address space, commonly on a network connected computer or server. When an application uses object-oriented principles, RPC is called remote invocation or remote method invocation. [39]

In terms of prices, most wireless broadband operators are now offering "flat rate" subscriptions (unlimited data volume subscriptions). However in practice, all of these operators have restrictions on data usage, for example capping the "flat rate" subscriptions to a total traffic of a few Gigabytes per month. So a user with a large data transfer requirement could encounter problems with such a subscription. Although most operators offer other subscriptions to high data volume users, these come at a higher price. However, the connectivity required for data synchronization has a very different characteristic than voice or other real-time services: because data synchronization is frequently delay-tolerant, and does not have a strict minimum bit-rate requirement, as a result non-real-time synchronization can utilizes the unused capacity of a cellular network –hence this traffic need not interfere with the operator's voice traffic. If there is a need to synchronize large files to which there have been a lot of changes, this will require either high data rates or a longer period of time for the synchronization. Today online real-time synchronization which requires large volumes of data to be transferred are not widely used. In contrast, when the amount of data that needs to be transferred is small, then real-time synchronization (even when it involves small amounts of change to large files) is very useful today. Due to increasing peak rates and the delay-tolerance of most cases of data synchronization source and target devices can opportunistically use the available capacity of the network to perform data synchronization in practice within acceptable time periods. When wide area cellular networks do not have sufficient resources, users can make use of wireless local area networks in hot spots to get connectivity.

2.7.2 Summary of real-time synchronization

Real-time synchronization can be the best way to maintain data consistency, but it requires continuous connectivity between source devices and target devices. Both target and source devices may have to do more computational work than for non-real-time synchronization mechanism. Real-time synchronization faces problems when the user experiences high latency due to limited available network capacity (as a result of competition with other users for these capacity or poor link path properties). Therefore, real-time synchronization is currently only used for synchronizing small amounts of time sensitive data, such as e-mail, PIM schedule or task data. In contrast, non-real-time synchronization is widely used. In the next chapter we will examination the design and implementation of a non-real-time synchronization system that will subsequently tested in our test bed.

2.8 Summary of performance evaluation

Performance evaluation is always an important matter for software. The design of a high performance synchronization process depends on the six factors mentioned in the following: data transmission load; end device computation load; number of devices to be synchronized; robustness of communication between the source and target device(s); memory usage on the end devices; and time complexity of the whole synchronization process. This chapter has examined these factors to offer some background for the rest of the thesis, especially these evaluations of the system that was implemented.

These factors always depending on each other, for example, given a specific amount of data to be synchronized, if the synchronization process minimizes the amount of data to

be transmitted, then fewer packets (and bytes) of data need to be exchanged potentially lowering both the cost of the synchronization and lowering the amount of time required to perform the synchronization. However, in order to exchange the same amount of data, the end devices may need to perform more power on computation in order to compress the data to be exchanged. If it is necessary to locally store two copies of the file to compare them (as in the case for the binary file synchronization examined above), the memory requirements may exceed the available memory of the device. Therefore it is better to design a synchronization protocol based on the specific resources of the end devices, computational ability, data rate of the connection, and the synchronization data types. The next two chapters will provide some background and motivation for synchronization solutions to be used for SQLite Database.

3 Database synchronization with a repository

There are many synchronization protocols and synchronization application designed for different use-cases. For example, to synchronize PIM data, SyncML is very suitable as it was designed explicitly for this type of data. For PostgreSQL, there is a database replication solution called Slony-I [20]. It is a trigger-based database synchronization solution specifically designed for PostgreSQL. This solution works asynchronously and provides database synchronization. The Slony-I usage scenarios are: database replication from a central office to branches in order to reduce network traffic or to speed up database transactions; database replication for load balancing, and hot backup using a standby server or upgrades to a new release of PostgreSQL [21]. For MySQL, there is solution called MySQL Replication [22]. It automatically and continuously replicates databases between two MySQL servers via an IP network. This could be used to provide database redundancy. For example, if a failure happens on one MySQL server there is a hot standby replicate on the other MySQL server. This solution utilizes a master - slave relationship between the two databases. The database that is being accessed and modified is the master. The slave server requests copies from master. Moreover, a master database can have more than one slave, and each slave database can be considered as a master that may in turn offer replication to this slave's database itself, therefore one can construct a tree of master – slave databases. MySQL Replication runs two threads on each slave, one is an I/O thread and the other one is an SQL thread. The I/O thread receives events, which are generated when operations occur on its master database (these can be extracted from the binary logs or transaction logs of the master database). The I/O thread writes these operations into its own log. Subsequently the SQL thread then reads this log and executes the events as updates to the slave database. The two threads work independently. In order to communicate between slave and master database, the slave database has to have an account on the master database. The exchange of data will resume at this point which the slave database periodically connects to the master and locates the position in the master's binary log where it left off the last time it connected. [23]

The database synchronization solutions mentioned in the previous paragraph always need a Database Management System (DBMS) is properly installed in both target devices and source devices. And it requests continuous connections between the target and source devices. If usage scenarios do not provide a continuous connectivity, do not have a DBMS installed in either source devices or target device, or the usage scenarios need a non-real-time synchronization, we would need to design another database synchronization structure. Later when we describe our tests, we will motivate our own design for a non-real-time synchronization protocol. To help with this design process we will analyze every stage during the synchronization process step by step and compare different potential solutions and evaluate different combinations of synchronization solutions. Finally, we implement a synchronization solution for our test case. Our design is based on zero configuration databases; in this test we use a SQLite as an example database management system.

3.1 Introduction to zero configuration databases

Considering synchronizing records in a database, the choice of DBMS plays an important role in the selection of an appropriate synchronization mechanism. We chose a light weight DBMS solution in order to suite the use case requirement.

Today more and more devices make use of a light weight DBMS. One form of such databases is called a zero configuration DBMS. A "Zero Configuration DBMS" means that the database does not need to be installed before it is used. Additionally there is no setup procedure and no server process. There is nothing needed other than telling a "Zero Configuration Database" is running [24].

Table 1: Comparison between the regular DBMS and "Zero Configuration DBMS"

	Regular Database	Zero Configuration Database
Install step	Must first install	No
Configuration step	Must configure the Database management system	No
Server process	Must start a database management system process running in the background	No
Database configuration	There is an initial database configuration process	No
Database permission	Database permission control must be specified in the configuration files	No
Database logs	Database transaction logs taken care of DBMS	No
Crash recovery	There is a crash recovery mechanism controlled by the DBMS	No
Troubleshooting	There are troubleshooting procedures when the database has problems	No

According to the web site http://www.sqlite.org/about.html, SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite is a self-contained, server-less, zero-configuration, transactional SQL database engine. It does not have a separate server process. SQLite reads and writes directly to ordinary disk files. The disk file includes tables, triggers, and views. In addition, the database file format is cross platform, hence it can be read both under the 32-bit and 64-bit systems, on both PC platforms the Apple Mac platform or other handheld devices OS. These features make SQLite a popular choice for an application file format. SQLite cannot be considered as a replacement for MySQL or Oracle, but it could be considered as a replacement for a "fopen()"[10] function. SQLite requires no maintenance from a database administer. In addition, SQLite can be used for small or medium - sized websites. Lastly, SQLite is fast and it requires no setup, hence SQLite can be used as a surrogate for an enterprise relational DBMS (using a subset of the enterprise database and subset of the transactions) and as an operational database in many applications settings.

Database transactions require that all changes and queries execute in an Atomic, Consistent, Isolated, and Durable (ACID) manner. SQLite implements serialization

[10] fopen() is function opens a file as indicated by parameters and returns a stream associated with that file.

transactions that follow ACID, even if the transaction is interrupted by a program crash, an operating system crash, or a power failure to the computer. [25]

SQLite has been used in many consumer electronics devices digital treatment devices, for example, Polar's Heart Rate Monitor [26] and many other devices. Implementing the proposed repository based on synchronization solution can provide SQLite database data consistency between separate SQLite databases on different machines.

The entire database must follow the transaction semantics. Although transaction logs are always maintained by DBMS, there is no DBMS running behind the SQLite database, hence there are no transaction logs for the SQLite Database. This means that we cannot base a solution upon processing a transaction log, but instead, we can observe all of the operations on the database and process those transactions completed to create self made synchronization events.

3.2 Introduction to the testing bed structure

This testing bed was designed for testing non-real-time synchronization system. The database records will be synchronized between two target devices and one source device. These three devices connect to each other via three separate TCP connections. Figure 16 shows the structure of this testing bed.

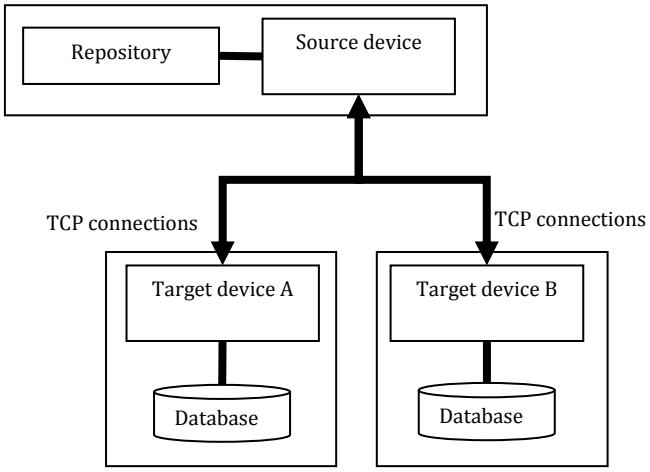

Figure 16: Testing bed structure

As Figure 16 shows properties in this testing bed structure, one source device has connection with two target devices. The source device has a repository attached with it, and both two target devices have database attached with them. This use case focuses on a non-real-time synchronization SQLite database over TCP network. As the chapter 3.1 mentioned, SQLite database would not provide any build in synchronization and transaction logs mechanism. Additionally, because of the characteristic of zero configuration DBMS, SQLite database on different machines would run separately, hence, separate SQLite databases would not have any build-in coordination mechanism in order to make data consistency between separate SQLite databases. In terms of data synchronization protocols, as the thesis mentioned in chapter two and chapter three, the widely used SyncML protocol is good for PIM data. In this testing bed solution, we would

not have any data synchronized related to PIM data. Therefore, according to these two facts from this use case, we have to build a database synchronization protocol on top of SQLite DBMS. And this synchronization mechanism would be based on the theory of repository mechanism in version control systems.

Table 2: Technical details of the three machines

	Source device	Target device A	Target device B
Name	HP WorkStation xw6000	ThinkPad X60	ThinkPad X300
CPU	Intel® Xeon® ™ CPUs 2.80GHz *2	Intel® Core Duo® T2300 1.66GHz	Intel® Core Duo® L7100 CPU 1.2GHz
Memory	2.0GB	3.0GB	2.0GB
Hard Disk	5600 per second	5600 per second	solid-state drive (SSD)
Network interface	100 Mbps Ethernet	100 Mbps Ethernet	100 Mbps Ethernet

Table 2 shows the technical details of three machines in this testing bed. There are two target devices connected to the source device via TCP connections. Both target devices have a local SQLite database. As can be seen in Table 2, the source device does in fact have more computational power than the target devices.

In the testing, we assume the structure of the database shown in Table 3. In this sample table, the "Items ID" is the primary key for this table. Both "Products Name" and "Selling store" is TEXT data type. And "Price" is "FLOAT data type".

Table 3: sample table used for testing bed

ItemsID	ProductsName	Sellingstore	Price

In this thesis, the purpose would be design, analysis and implement a synchronization system for this use case.

3.3 Synchronization solution analysis

We begin the analysis by dividing the complete synchronization process into different steps. After that, we will analyze each of these synchronization solutions step by step in order to design a protocol.

3.3.1 Key steps in synchronization protocols

Synchronization protocols would have the following mechanisms:

- Synchronization protocols must have mechanism to identify the changes in the data to be synchronized (in this case, records in SQLite database that have changed).
- Encode these modified changes from changed records into a message in order to exchange the message between a source device and target devices.
- Read the messages and decode in order to merge the changes with the local record in order to bring the two databases into the same state.
- Handle conflicts, to recognize different devices in order to identify which changes are coming from which devices.

- Error handling mechanism and logging system in order to have greater robustness and to understand what, when, and how changes have been made.

Table 4 below shows general requirements of synchronization protocols for use with databases.

Table 4: Requirements of a synchronization protocol for use with databases

Requirement	Priority
1: To identify the records that have been changed	Must
2: To encode the changes in messages	Must
3: To transmit and receive these messages	Must
4: To handle conflict situation	Must
5: To recognize different devices	Must
6: Error handling mechanism	Should
7: Logging system	Should

As described in the previous chapters there are basically three ways to identify modified records in a database, using timestamps, synchronization flags, or keeping a transaction log. As SQLite does not keep a transaction log we cannot use this method. Using timestamps consumes more storage than using synchronization flags, but enables some additional features, which is not request in our testing case. Since according to the chapter two, only synchronization flags mechanism does not multiple devices process synchronization, and using timestamp would have more storage consuming, thus in our testing bed, we implement a new mechanism to handle changes identifications. In this mechanism, we add an additional field to each record to be used for a synchronization flags. Rather than using a “bool data type” to identify records that have been modified we have used, an “INTEGER data type” field. Since “INTEGER data types” can store more than a single bit, we have used these additional bits to offer additional functionality; for example, we will keep track of the type of changes insertion, modification, or deletion. We will also use this integer field at the source device to encode which target devices modified the record. This field enables us to meet the first and fifth requirements listed in Table 4. Since there are two devices in the testing bed, we will encode the device's ID as one decimal digit and encode in another decimal digit the operation that was performed locally to change the record (i.e. modification, deletion or insertion). Figure 17 shows these two encoding mechanism.

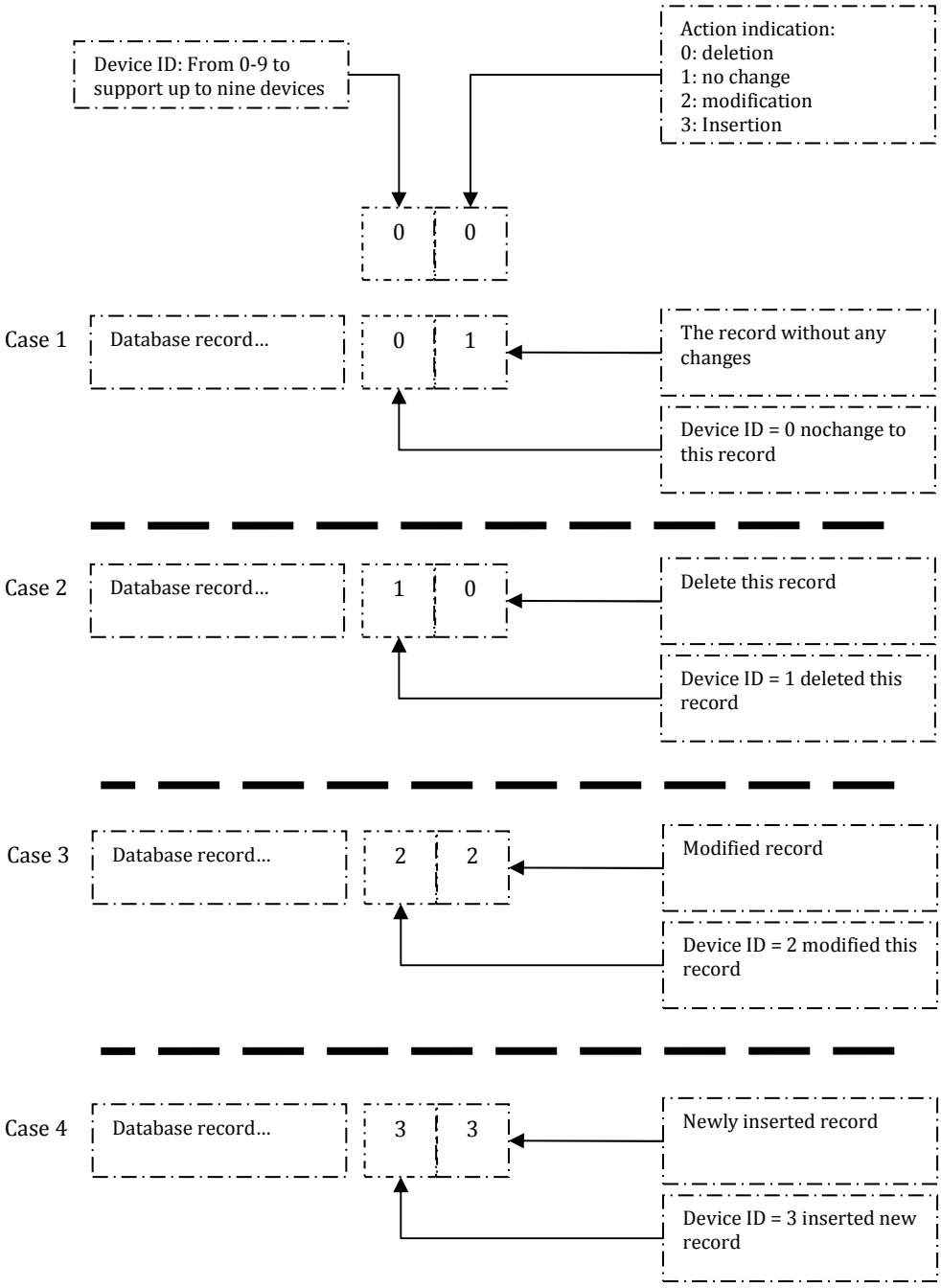

Figure 17: Indication different actions of database record

Figure 17 shows how the synchronization field works. These two digits indicate both the devices and action performed. For example, case 2 shows device 1 deleted a record; case 3 shows device 2 modified a record. Case 4 shows that device 3 inserted a new record, which the device does not involved in our testing cases. Using INTEGER data type and this mechanism would have less storage consuming than timestamps solution. Additionally, it would be available functionality to monitor database operations of every activity.

After the protocol data structure has been designed, an additional field is introduced in to the database table. Table 5 shows a table structure used in the synchronization protocol

Table 5 Table structure in the synchronization system

ItemsID	ProductsName	SellingStore	Price	SyncFlags

Now this table structures support identify changes and mark the device making the changes, and in the Figure 17, it describe how these fields values should be used. Following this design, in order to process the synchronization fields, the process has to query the entire table. As the table increases this query time will increase. One way of avoiding this quarry time is to great another table called a *Sync Table* to store a list of the records whose synchronization flag has been changed, the purpose of this utilize is to self construct a transaction log. This log can be done by adding a trigger to all the tables, such that every time a record is changed and the trigger will be invoked and it can generate the transaction entry. Note that the transaction log needs only include the primary key, as all of the needed information is stored in the actual record. In this way, the synchronization system can simply read the *Sync Table* and process the records to know which records have been changed, how the records have been changed, and which device changed the records. The *Sync Table* can be implemented based the structures shown in Table 6.

Table 6: Sync. table structure

Sync ID	TableName	RowID	SyncFlags

In the Table 6, the first field is sync ID which is the primary key of this table. Next is a table name indicating which table should be synchronized, also means which table was changed before, next is the record identifier (called RowID) which indicates which record in the named table was modified - in our case this field contains the itemsID as this is the primary key in our example table structure (as shown in Table 5). The final field is the syncFlags which is a copy of the sync flags for the indicated record. Therefore, at run-time, the system needs only walk along the Sync Table and process the records of the database changes. Note that it must also access the actual records in the database as it needs to update the synchronization field of the record itself, once this record has been synchronized with the other database(s).

3.3.2 Synchronization protocol analysis

The next step is to encode the changes into messages in order to send them to the other device(s). This could be a textual encoding or a binary encoding. There are advantages and disadvantages to both. They concern the following factors:

- End devices computational cost
- Data transmission cost
- Synchronization process latency requirements

- Protocol functional extension
- Time complexity during synchronization
- Error prevention & control

In terms of the end devices, computational cost, both methods of encoding (i.e., textual versus binary encoding) share the common process of having to walk along the *Sync Table*, process the data in the *Sync Table*. The difference will be in encoding the changes into textual or binary records. Since in our testing bed case, the fact is both the source device and target devices can be considered to have powerful computational capacity. According to the static testing regarding the time consuming of fill in amount of data in to the testing table and after that we have a program encoding this data into SQL statement textual format or SQLite binary format below which is shown in Figure 18. This test was done on one of the target device, which is ThinkPad X60.

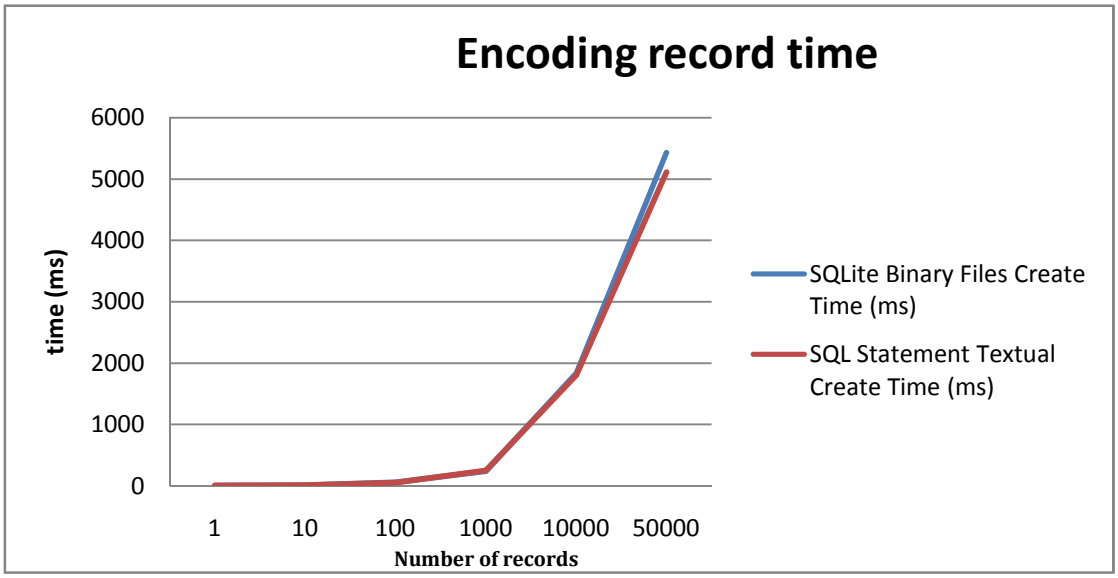

Figure 18 Encoding records time to either SQLite binary files or SQL Statement textual files

In Figure 18 shows that the time consuming of encoding amount of data increases along with the number of data increases. But there is not much difference between encoding data to SQLite binary file and encoding data to SQL statement textual file. It means that end device would have almost the same amount of computational cost consider the SQLite binary files or SQL statement textual files. Therefore, according to the statistics, if only device computational cost is considered, either choosing SQLite binary files format or SQL statement textual files format, it is not a big issue for the entire synchronization system.

Considering the second factor, end device data transmission cost, this factor is main affected by the file length of exchanging messages between source and target devices; bandwidth between source and target devices and other minor issues. In this thesis, we firstly take the file length of exchanging messages into consideration. As pervious paragraph mentioned, the exchange messages sent between end devices can be encoded into SQLite binary file format and SQL statement textual file format. So in this part, we will compare the data length of a file when it is encoded into either a SQLite binary

format or SQL statement textual format. For this textual format, we choose the SQL statement format (which is presented as a *.sql file in the file system), and for the binary format we chose the SQLite binary files format (which is presented as a *.s3db file in the file system). We compare the files size between those two formats for a number of records filled in the testing table in SQLite Database (The table structure was described in Table 5). The results are shown in Table 7. Note that the text fields in these sample records were generated by generating sample record which is shown in the Appendix Sample Data.

Table 7: Files length (in bytes) comparison with SQL statement textual files encodings and SQLite binary files encodings

Number of records	Files size (KByte)	
	SQL(Text)	SQLite (Binary)
100	7.4	6.1
150	9.3	7.1
200	11.4	8.1
250	13.4	9.2
300	15.3	10.1

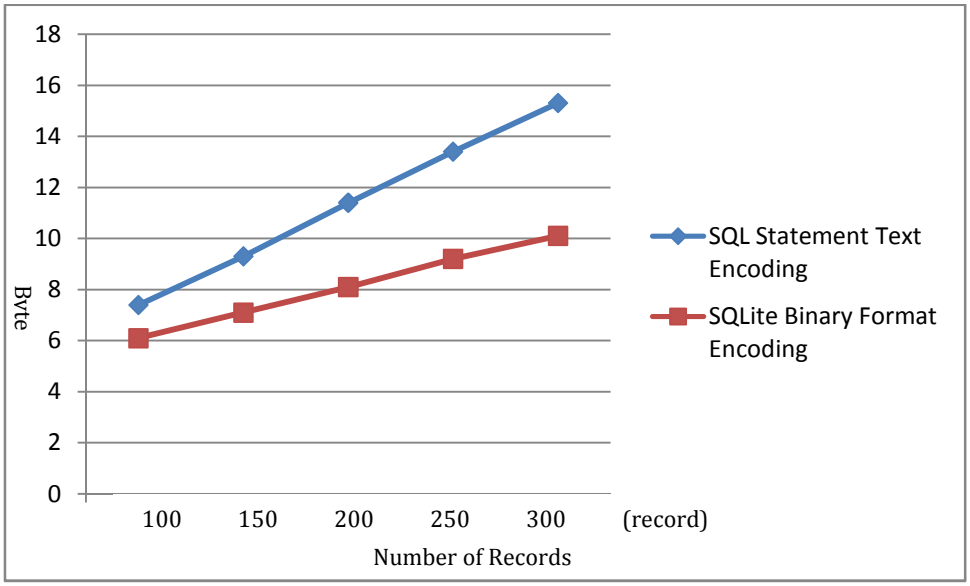

Figure 19: Files length comparison SQL statement (text) and SQLite binary format

In Table 7 and Figure 19, with 100 records in the table, the textual SQL format required 7.4 KB, but the binary encoding required 6.1 KB. As the number of records increase, the binary format continues to increase its advantage over the textural format. Before making a final decision about which encoding to use, we also need to consider the rate at which records are added to the database or the size of the data for updates. According to this statistic, for small amount of records, it means the number of records less than 100, there is not big difference in terms of files length after encoding records to either SQLite binary file format or SQL statement textual file format. Note that the size of deleted records is not a concern, since only the information needed to identify which record is to be deleted needs to be transmitted. Thus, for some use cases which need frequency

synchronize records between target device and source device, it would not decrease the transmission cost a lot either using SQLite binary format encoding or SQL statement textural format, but regarding synchronizing large number of record, using SQLite binary file format would have lower transmission cost between target device and source device.

With regard to transmission latency binary encoding of the data can utilize a smaller number of bits than textual encodings. Less data to transmit directly reduces the transmission time. More specific, we can estimate the compress factor for binary versus textual encoding by comparing uncompressed to compressed text using a number of different compression algorithms. Typically the compressed file will be around 60% smaller than the uncompressed file. [27] [28] Therefore using binary encoding would allow transmission to be around 60% faster than textual transmission. It is not certain that in a given situation we will experience this roughly factor of two decreases in transmission time, as we must also consider how these messages are processed by the transmission protocol and the application layer protocol. Experience with many internet protocols has shown that the advantages of the textual encoding (simplicity, readability, etc.) generally favor textual encoding over binary encodings –unless there is a very large volume of data to be transferred.

Consider the next factor, the latency of synchronization process. In this thesis, we are planning to analyze the synchronization latency issue based on the following module. In this module, we are going to fill in amount of demo records into the sample table, which is according to the Table 3, based on different time interval on a client device. After that, those demo records will be synchronized with a synchronization server at a specific time period. After the synchronization process, the demo data would arrive at synchronization server with a SFTP transfer time. This testing case works like the block diagram in Figure 20.

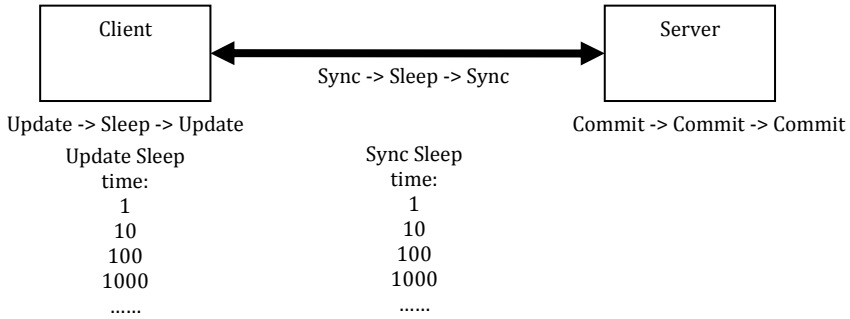

Figure 20: Latency testing cases

In Figure 20, we have client side application fill testing table with sample record automatically, and each insertion action would have a sleep time, the sleep time would be from one time unit up to one thousand time units. Additionally, synchronization takes place at a specific time interval, which would be from one time unit up to one thousand time units, as well. In this case, we will take a look at the file length to be transferred and SFTP file transfer time. And in this case, the total latency would be the SFTP transfer time plus end device encoding and decoding time. Figure 21 shows that the client has the longer sleep time the shorter file length will be.

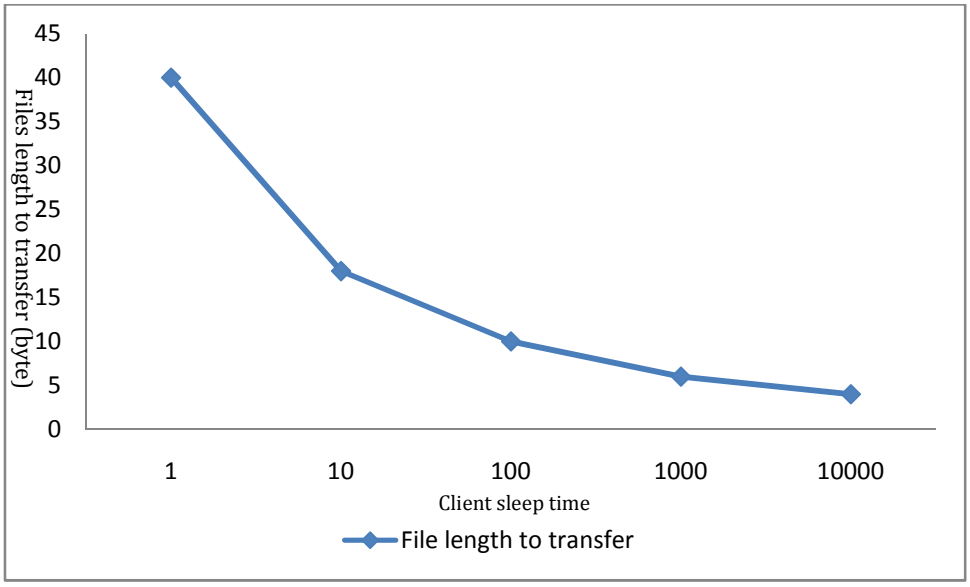

Figure 21: File length to be transferred based on different time interval

Since this case works with 100 Mbps Ethernet connections between client and server, according to the Figure 22 and Figure 23, there is not much variation in client side application working with different specific sleep times and synchronization sleep times.

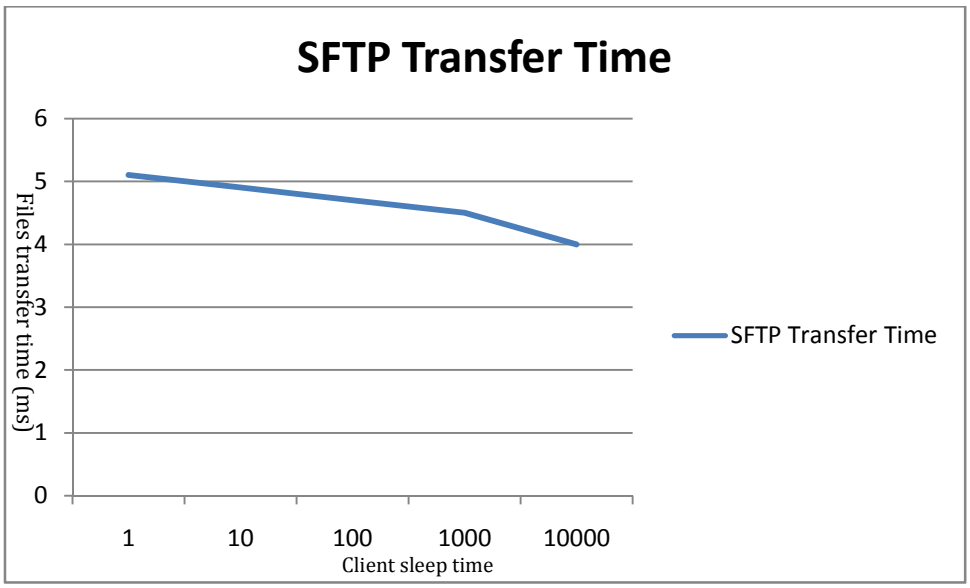

Figure 22: SFTP transfer at client has sleep time interval

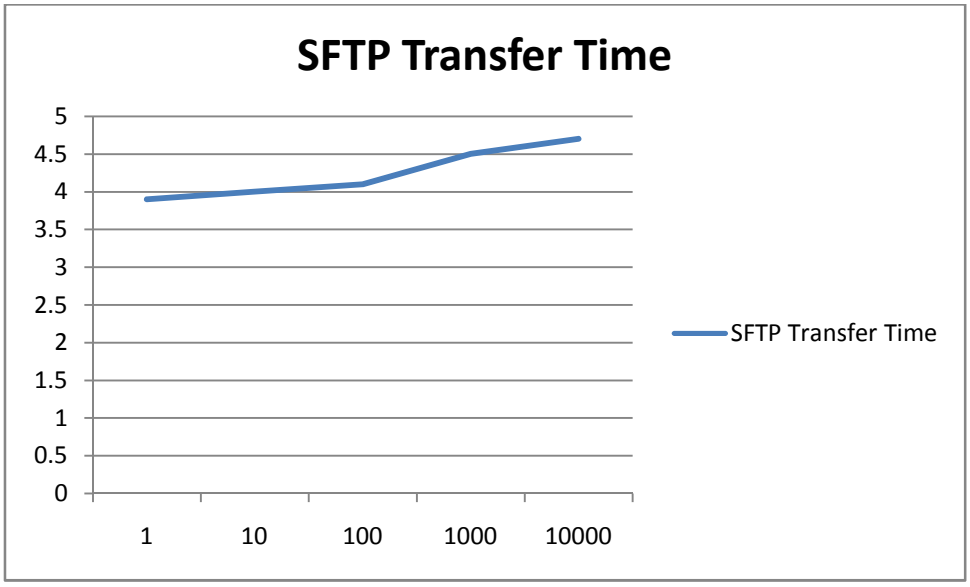

Figure 23: SFTP transfer time when synchronization process has sleep time interval

Therefore, according to Figure 22 and Figure 23 end devices latency in this case can be considered very low.

Next factor, we will consider the functionality of the protocol. One of the important issues would be if this protocol will be easy to add additional functions or to change functions. In order to extend more functionality in this protocol, the import part would be change the structure of the current table. As it mentioned in Chapter three SQLite binary formats would present all table structure in one binary file. Therefore, there would not be additional work to modify table structure in current solution. So it means that more functionality can be added in currently protocol.

After that, we would address with time complexity of the synchronization protocol. According to the features of synchronization protocol, the time complexity of this protocol is dependent upon the end devices computational capabilities plus the SFTP data transfer time between a source device and target device. Based on the Figure 18, the end device encoding time is not increase significant in different volume of record. But there would be a significant overall difference in the synchronization time until the number of messages becomes quite large or the contents of each message are very large (for example if the database contains large high resolution images). However, the overall time complexity would be considered as a linear equation function $T(n) = O(f(n))$.

Finally, regarding with the error prevention and error control, textual encoding solution is generally more robust to bit errors during transmission. In contrast binary encodings are more sensitive to bit errors, especially when there is a very high degree of compression. In this case a single bit error can destroy the whole message. However, as we will use an underlying transport protocol, it is important that error detection, error correction and retransmission functions the transport protocol will handle. For applications such as databases it is very important that the databases remain consistent, so we must detect errors even at very low rates - otherwise after some time the databases will diverge - unless we make a complete new copy of the database.

Based upon the analysis above, we use radar charts to show the advantages and disadvantages of sending or textual encoding of our messages.

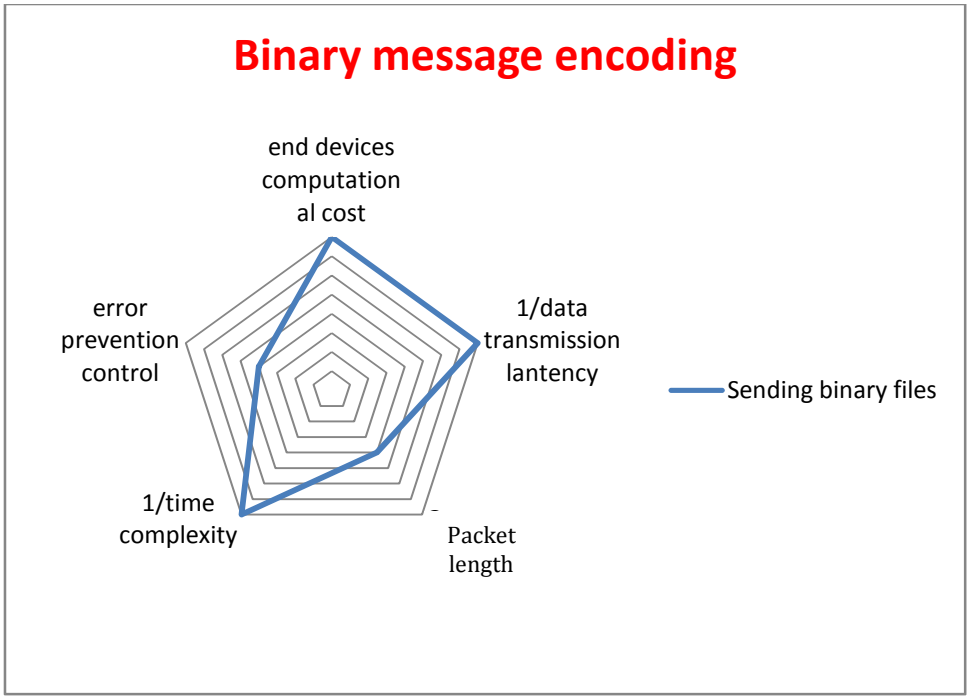

Figure 24: Performance of binary message encoding

As shown in Figure 24, using binary encoding offers lower transmission latency, lower time complexity, at the cost of computational effort, and protocol flexibility.

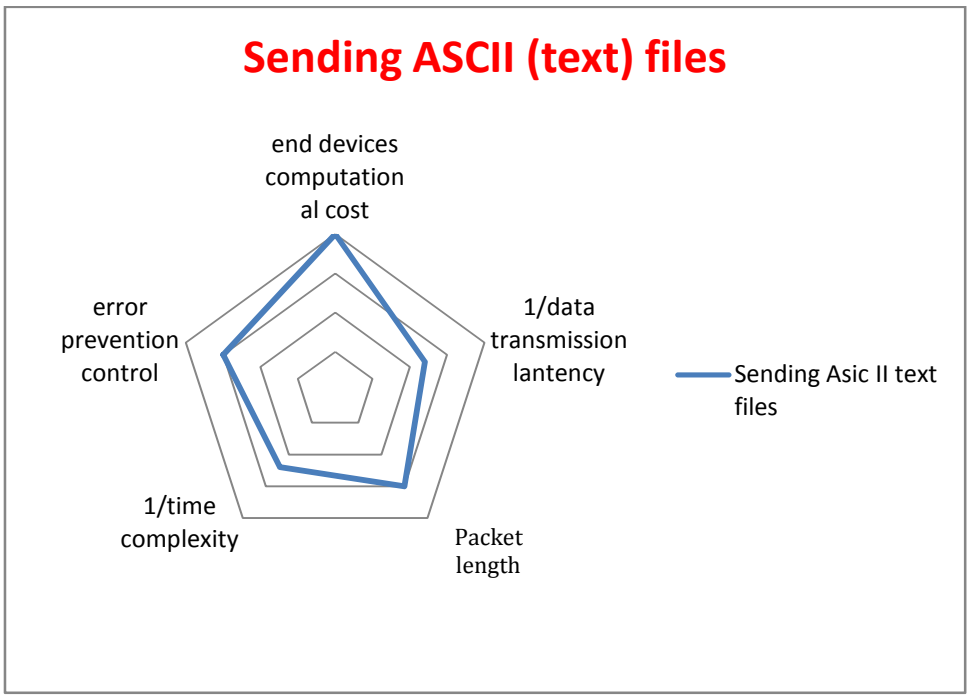

Figure 25: Performance of textual message encoding

In contrast Figure 25 shows that although using textual encoding messages between source devices and target devices has better error handling ability and greater protocol

flexibility, the transmission latency and time complexity are higher than encoding message to binary. So for this database synchronization protocol it is better to send SQLite binary format messages between source devices and target devices. It should be noted that if there is a very large amount of data to be transmitted, then it could be sent as binary data using another mechanism - that offers compression, for example, using a transport protocol shim layer that would perform compression on the data stream.

4 Implementation a non-real-time synchronization system

This chapter describes an implementation of a non-real-time synchronization system for a use case of a telemedicine system. The mechanism of this non-real-time synchronization is based on the analysis and performance evaluates part in the previous chapter. This system runs on top of a Microsoft Windows platform. This implementation will mainly be used to verify the design that we made in the previous chapter. As noted earlier, we have designed this synchronization system to handle the use case of zero configuration databases, specifically SQLite. Additionally, since this commercial system is running on top of a Microsoft Windows platform, the implementation of the synchronization system is based on the .NET framework. The source and target devices are assumed to have a TCP/IP protocol stack. And both of source device and target devices would have 100 mbps Ethernet connection. The synchronization protocol will be based upon the concepts described in the previous chapter.

4.1 Synchronization system overview

The purpose of this synchronization system is to provide SQLite database running on different machines having data consistency. All different machines are connected via TCP connections over underlying IP network. The physical network could be an Ethernet, 3G cellular network, or any other network that can provide IP connectivity. We will also assume that there will always be new data arrived at target devices. This synchronization system will determine the record differences on the SQLite database s on both source device and target device and perform synchronization in order to ensure *eventual* data consistency. It is important to emphasize that the goal is only eventual data consistency, as we want to realize a non-real-time synchronization system for zero configuration database in order to avoid with each other requiring that the different machines always have connectivity to all other machines. A system structure overview is shown in Figure 26.

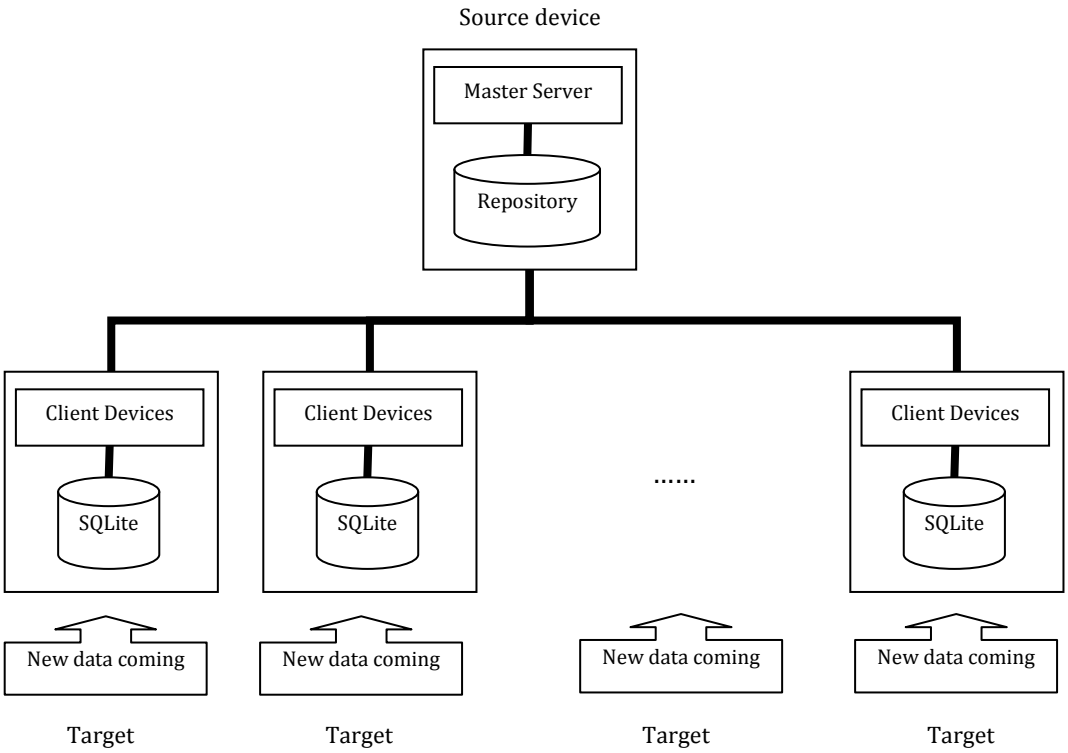

Figure 26: Synchronization basic structure

In Figure 26, there is one source device, which can be considered as a synchronization server, and it implements a central repository. This synchronization server communicates with a number of target devices; these will be called synchronization clients. Changes always occur first to the client database. Each synchronization client has a local SQLite database. Synchronization is used to make all SQLite databases have data consistency. In our scenario there are two client devices. Figure 27 shows a data flow during a synchronization process in more detail.

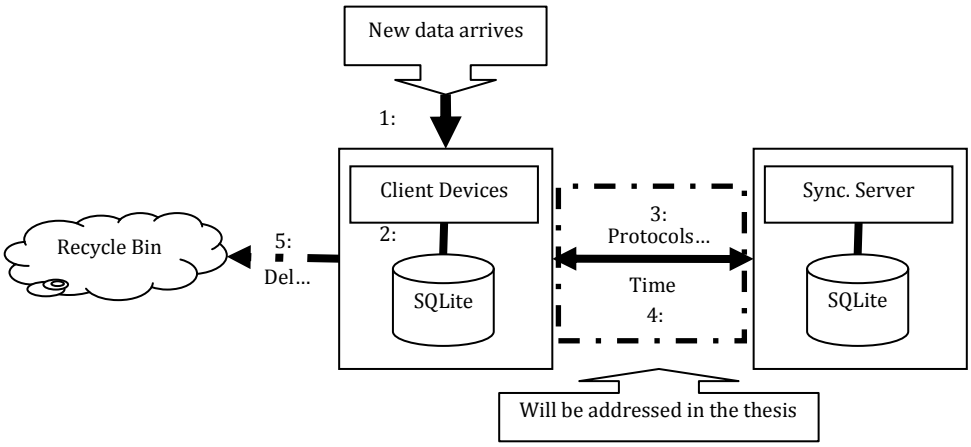

Figure 27: Synchronization data flow

In Figure 27, 1: New records come from client's I/O devices; 2: Client writes record to its local SQLite database; 3: client sends messages with new record to synchronization

Server; 4: The new records could be sent in real-time or queued for sending later (i.e. non-real-time); 5: Client deletes old record from its local database. We assume that initially both client and server have data consistency. Next event number one happens with the arrival of some new record to the client's database. Next, the client device writes the new record to the client's local SQLite database. The local SQLite database changes based upon this new record. The format of this record is assumed to be plain text. Then the synchronization protocol takes over in order to update the other databases by sending messages between the server and the other client(s). These actions are events number three and four. If a local record is deleted, then the deleted record will be sent to a recycle bin after the client synchronizes with server. It is important to note that the local records will only be deleted after clients have synchronized with the synchronization server, so that no new data would be lost - even if it only exists as a valid record for a short period of time.

In this case, testing records to be synchronized are stored in a table called "Comment". In order to keep the testing case sample, we only keep three columns in the testing table. The table structure is shown in Table 8.

Table 8: Structures of Comment table

Row name	Data type	Comment
ItemsID	INTEGER	Primary key
ProductsName	TEXT	
SellingStore	TEXT	
Price	FLOAT	

As Table 8 shows, there are four columns; they are "ItemsID" with "INTEGER" data type, "ProductsName" with "TEXT" data type, and "SellingStore" with "TEXT" data type and finally the "Price" with "FLOAT" data type. The primary key is "ItemsID" This table is stored in a SQLite database, representing as a "*.s3db" file on the disk.

4.1.1 System hardware and software configurations

This subsection describes the details of hardware configurations in whole system. Our testing bed consists of three physical machines, one machine is a synchronization server and the other two machines are synchronization clients. The synchronization server as described in "Table 2: Technical details of the three machines". The synchronization server connects via a 100 Mbps Ethernet interface to an intranet with an interface having an IP address of 192.168.0.20 / 24.

One client machine ($client_1$) as described in "Table 2: Technical details of the three machines" is connected via a 100 Mbps Ethernet interface to an intranet with the interface having an IP address of 192.168.0.10 / 24. The second client machine ($Client_2$) as described in "Table 2: Technical details of the three machines" on page 31 is connected via a 100 Mbps Ethernet interface to an intranet with the interface having an IP address of 192.168.0.11 / 24. The network topology is shown in Figure 28.

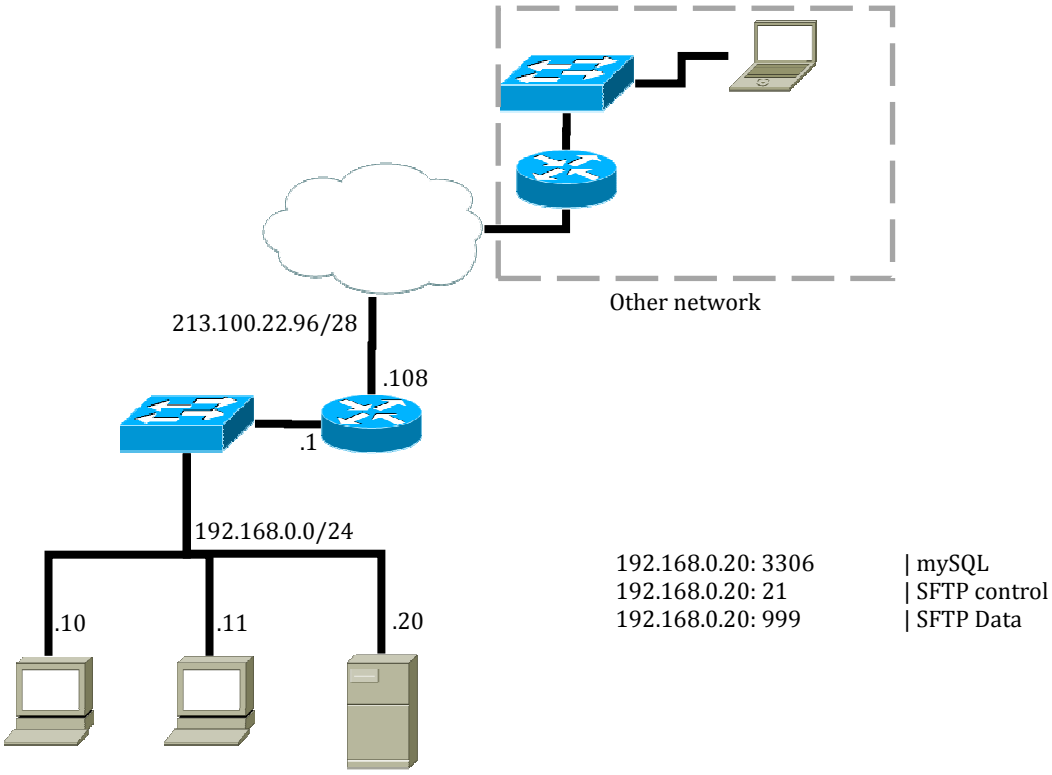

Figure 28: Network topology of the synchronization system

As shown in Figure 28, the three machines are connected to a local private network with the IP addresses: 192.168.0.10/24, 192.168.0.11/24, 192.168.0.20/24. The synchronization server provides two services: one is a MySQL database service and the other one is a SFTP service. Within the private network, the default gateway is 192.168.0.1. This is the address of a router with a public IP address 213.100.22.108/28. All of the interfaces in the 192.168.0.0/24 network are connected to a switch with supports non-blocking 100 Mbps full-duplex Ethernet connectivity.

Concerning with a software structure of a system, the synchronization system is to be developed based on .NET framework. All machines are equipped with .NET framework 2.0 the Microsoft Windows® Operating System (OS). Because the private intranet has internet connectivity, all of the machines in the private intranet maintain Coordinated Universal Time (UTC) [11]time using the same Network Time Protocol [12](NTP) server, which is "time.windows.com".

4.1.2 Synchronization solution introduction

For this non-real-time database synchronization, one way of performing SQLite synchronization is called "row–based" synchronization mechanism. In order to implement this mechanism, each row in the database tables that are to be synchronized must have a "DataUpdate" field which will contain a time-stamp in UTC format or a synchronization flag. In this case, we have chosen to add a field called "isValid"

[11] Coordinated Universal Time (UTC) is a standard time based on International Atomic Time.
[12] NTP is a protocol designed to synchronize the clocks of computers over a network. In this case we use the Windows system default NTP server "time.windows.com"

containing a synchronization flag. In addition, the "LastSyncDate" will be saved in a separate table. We have implemented an additional table called "syncTable" to put recent synchronization record. During the synchronization process, this system will process the following steps:

- First the program will read information from "syncTable" to learn the last time that synchronization was preformed;
- Next walks the "syncTable", this must be done in the proper order in order to prevent foreign key constraints, to find the rows that are not yet synchronized;
- Next, the rows which are not yet synchronized are sent to the central repository, along with their synchronization flags (the rows are read using the SQLite database API in .NET Framework);
- After the central repository receives messages from the clients, this server will insert all the records which are marked as unsynchronized. After the client has received notices that the server has completed synchronization of the data it will reset all the synchronization flags.
- If there are conflicts (for example, rows that changed on both server and client), the server has to resolve the specific conflict. For this resolution method we will rely on either a user interface to ask users which version to keep or use an algorithm to choose one of the versions.
- The server inserts or updates rows into the server's SQLite database and sends to the client the rows that have been changed since last its synchronization with the server.
- Finally, the client either inserts or updates these rows into its local SQLite database.

While row based synchronization is simple, it would need a complex process to manually configure and prone to conflicts and record. [29]

There are many different solutions using the same "row based" synchronization concept to handle database synchronization. One of these is called "Auto-Generated Update SQL". [30] This approach generates SQL statement and transmits these SQL statements between nodes in order to synchronize records. This solution works when the two tables have matching records, the primary table is updated with the data from the secondary table and if the secondary table has records that the primary table does not, then those records will be appended to the primary table. This solution provides synchronization without a hassle replication. [31]

4.2 Synchronization system implementation

This section describes an implementation of this synchronization system. First, we will analyze in detail the synchronization process on the current server itself and then decide a structure of this system, which is made in the previous chapter. The details that we did examine include how to handle the messages mechanism exchanging between a synchronization server and synchronization clients and how to identify the user.

4.2.1 Synchronization process

In order to perform synchronization, there are five questions: (1) how end devices identify database record changes. (2) How synchronization server and clients exchange

messages. (3) When the synchronization server and clients exchange messages. (4) What messages format the client and server will be used. (5) When the clients and synchronization server would receive record.

Regarding the first question, since SQLite does not have transaction log mechanism, we created a table called "syncTable" to log the SQLite database changes. In order to make this "syncTable" work, a trigger is set for all tables that are to be synchronized. The function of this trigger is to record all the database changes into syncTable. The table ID is a primary key of 'syncTable" table, additionally synchronization flags are saved in the syncTable. Figure 29 below shows a work flow of "syncTable".

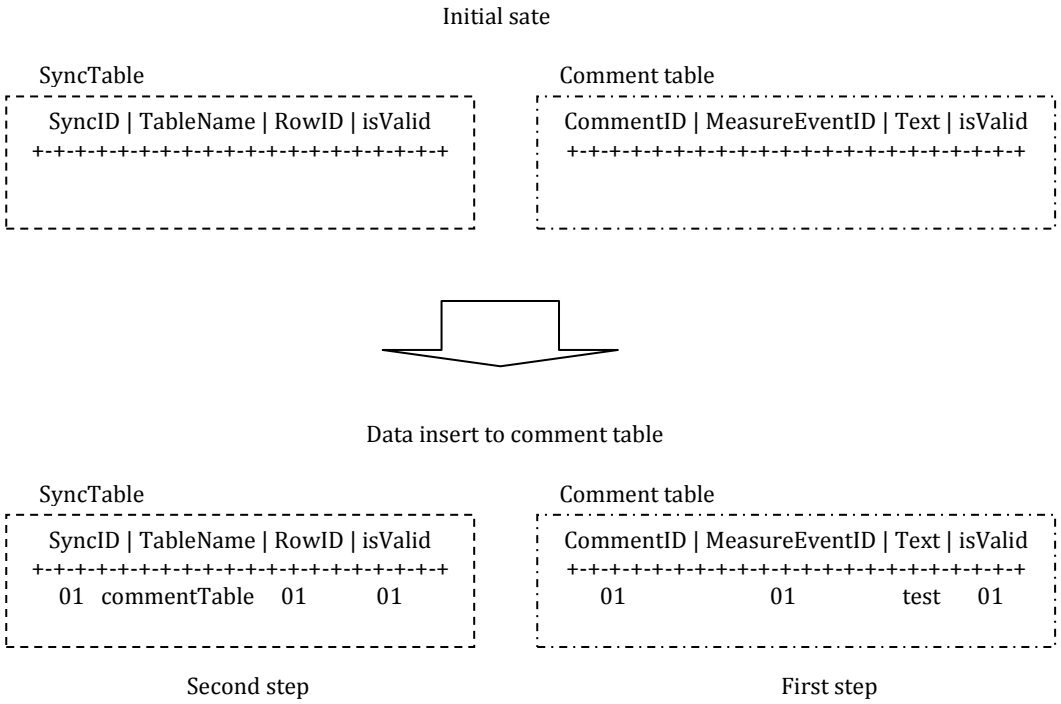

Figure 29: SyncTable work flow

As shown in Figure 29, at initial state, both comment table and syncTable are empty. When a record is inserted into comment table, the trigger will trigger an event which inserts the table changes into the syncTable. The "SyncID" is a primary key of the syncTable and is generated as a linear sequence. The table name is the table which had been changed and the "RowID" indicates which row changed in that table. The "isValid" field indicates that "syncFlags" in the comment table are to be used when synchronization process is recorded.

The next step, concerns exchange of messages between synchronization server and clients, because we are interested in a non-real-time synchronization system and since there is already an SFTP service running on the server, our synchronization system uses this SFTP service to provide reliable (and secure) data transmission between the server and clients. Rather than have the clients and server exchanging individual messages, the application start a synchronization process (for example when the application is started) or a user clicks a "Synchronize" button, then messages will be exchanged. To create the effect of exchanging lots of messages concerning the synchronization of individual

records, we create a table and send this table via SFTP. We made this design (since we found that the SQLite binary file was smaller than the SQL statement textual format and we can utilize the retransmission, error detection and error correction functions of the SFTP application. The cost of creating this file rather than sending the individual messages is that there is an increase in the latency between a local update and when the report target is updated. However, buffering increases the amount of data that will be transferred at one time, hence increasing the opportunity for compress and simplifying the operations. It also means that when the synchronization happens that the communication link can be turned on, while it can remain off at other times - potentially saving battery power). The receiver reads the SQLite database and performs an update with its local database. The SQLite binary files can be read by a standard SQLite API. Note that this same approach could be used to send a compressed version of the SQL statements, as SFTP already includes compression as one of its feature. We did analyze of this part in the previous chapter measuring the performance of the system in this testing bed.

4.3 Modules of the synchronization system

The synchronization system can be divided into the following modules: client authentication modules; data transmission modules; and SQLite database handler modules. Because, there is already an SFTP service and a MySQL database running at the server, in order to simplify the synchronization system, we do not plan to add additional services on the server, but rather the synchronization system uses the existing MySQL database to manage authentication record and exploits this SFTP server for data transmission.

4.3.1 Synchronization servers detail

The synchronization server performs client authentication; provides secure files transfer between server and clients; and synchronizes SQLite database(s) with separate SQLite database. Figure 30 shows server side of this synchronization system.

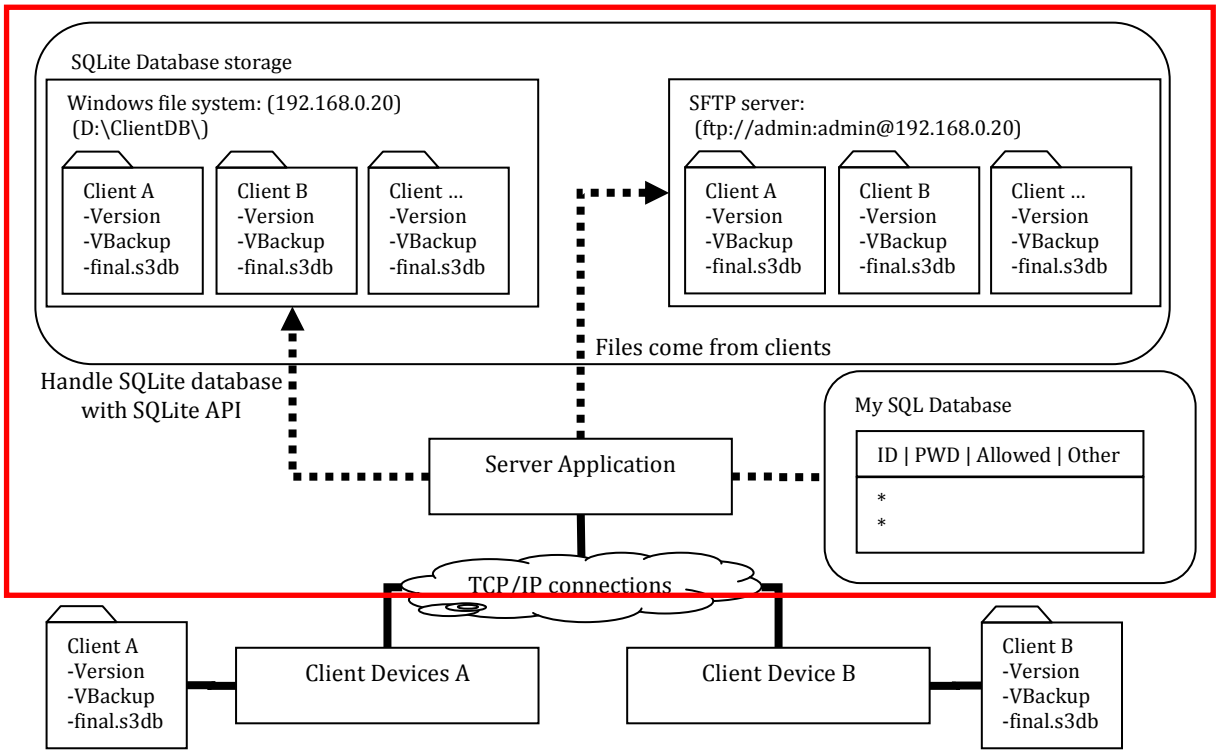

Figure 30: Overview of the structure of the synchronization system: Details of the server side

As shown in Figure 30, the synchronization server has three main modules. The first of these three uses a MySQL database to handle clients' authentication information. This is shown as a table with the columns: ID, PWD, Allowed, and Other. These fields contain user name (as the ID for the user), a password and a field that indicates what operations this user is allowed to perform, and other information about current user - respectively the MySQL database. The synchronization server utilizes an existing SFTP server to secretly send and receive files containing what are essentially database transaction logs. When one of these transaction log files arrives as a SQLite binary file at the synchronization server, the server application utilizes a SQLite API in order to read the SQLite binary file and complete synchronization process. As shown in the upper right of Figure 30, every client has associated with it a directory (folder) named with client's ID in the SFTP root folder. In each client's folder, there are two folders: "Version" and "VBackup". When each SQLite binary database arrives it will be placed in the "Version" folder. After the synchronization process, that SQLite binary file will be moved to "VBackup folder". Therefore every time synchronization occurs, the synchronization mechanism will check the "VBackup" folder first, to see if there are differences in the files. The "final.s3db" file is the server's copy of the client synchronized SQLite database file. Both the SFTP server and server files share the same file structure on the disk.

According to the Figure 30 shows the server side working flow during the synchronization process. Here we assume that the "client A" initiates synchronizing with the server. After logging in client device "A" connects to a synchronization server and requests synchronization. First, the client device must be authentication with user's credentials stored in the MySQL database. After authentication successful, the process

will move to the next step. As shown in Figure 31 block the synchronization folder on the client device will be granted access to the server's synchronization folder via SFTP.

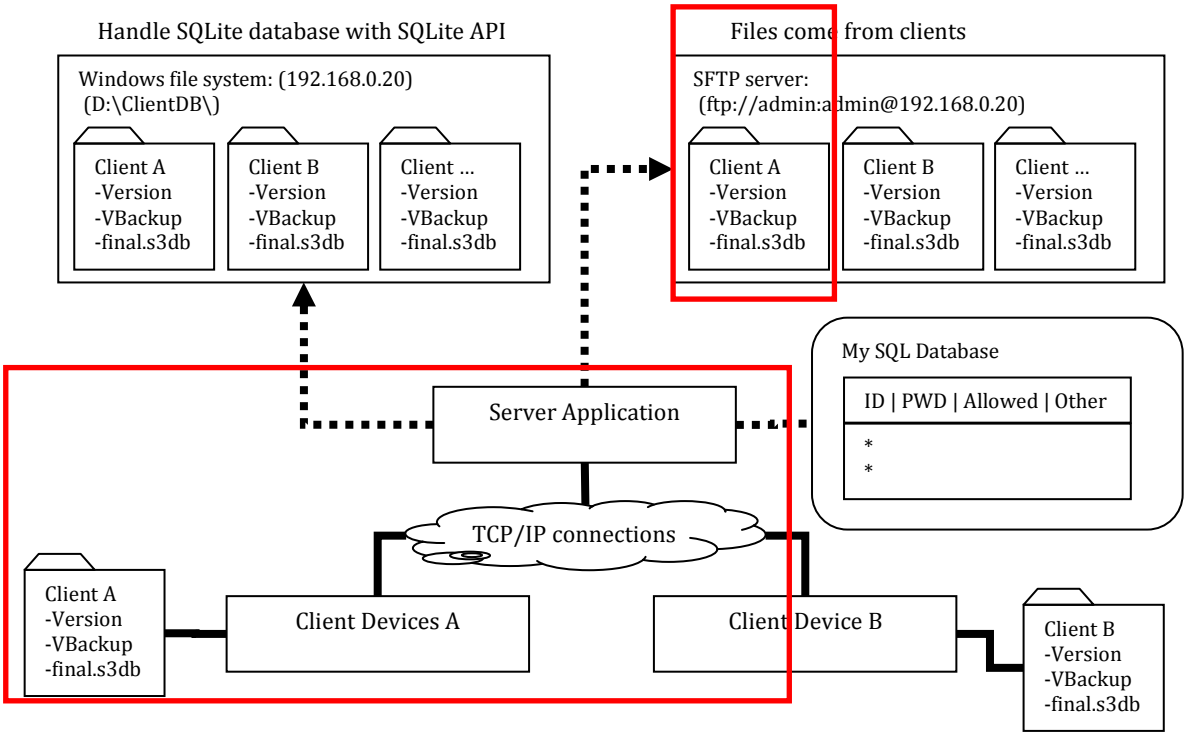

Figure 31: Synchronization server work flow to send files

Figure 31 shows the SFTP server has created a secure connection between client device A and its associated synchronization data folder. The FTP LIST[13] command will be used to check if there is difference between the client's "VBackup" folder and the server's "VBackup" folder. The missing files will be sent. After this step, as Figure 32 shows, if there are new files in the folder, then the synchronization server will utilizes about SQLite API to complete the synchronization process by updating and merging the received file with the local copy of the client's database.

[13] The FTP LIST command will return information about a file or directory from the current working directory.

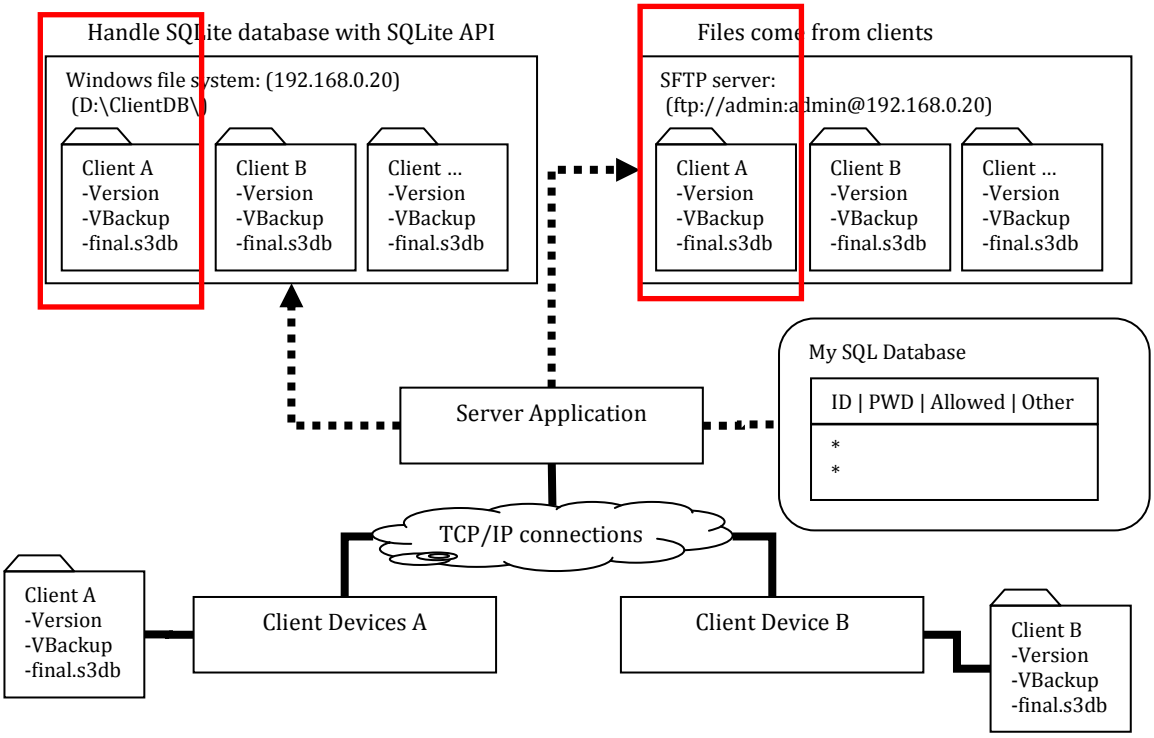

Figure 32: Synchronization server work flow for database synchronization

As shown in Figure 32 block, in the last step of synchronization the synchronization server application uses the SQLite API to insert or modify its copy of the client's final.s3db SQLite database files now the synchronization process is complete. This same process can be used to synchronize a client. Part of the C# code to implement the server is included as appendix C.

4.3.2 Client application in detail

The synchronization process on the client side shares the same concepts as the server side. Part of the C# code of the client side implementation is included as appendix C.

In Figure 33, we can see that the client requests a connection to the server then sends an authentication request to the synchronization server. After authentication successful, the SFTP server establishes a secure and reliable connection between the server and client. After this SQLite binary files will be sent to the client's synchronization "Version" folder. Next the client application will utilize SQLite API to insert or modify data in the client's local "final.s3db" file based on the contents of the SQLite binary file(s).

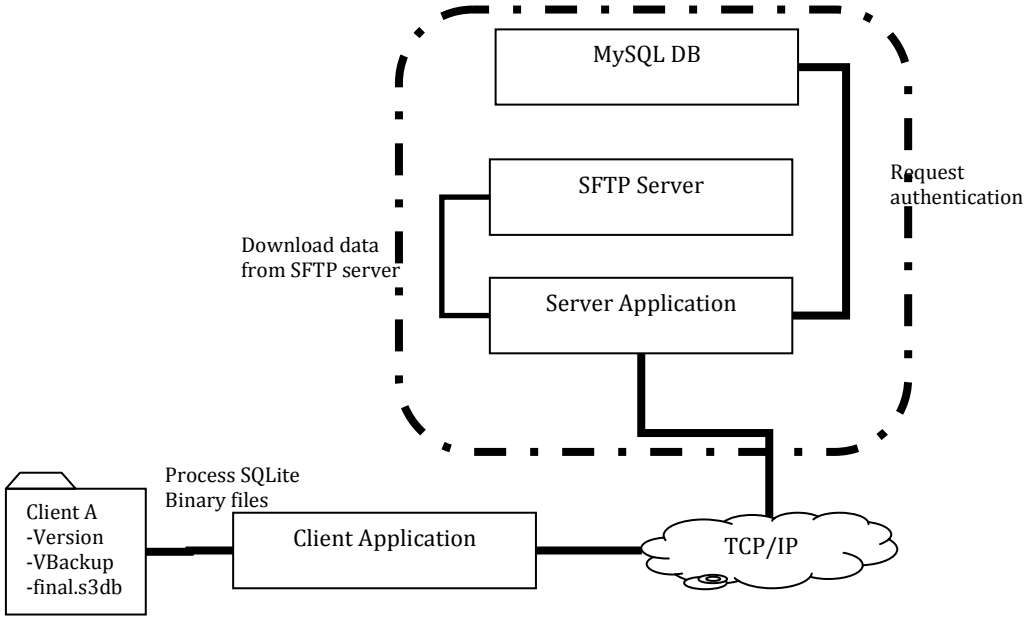

Figure 33: Working flow of the client side of the synchronization process

4.4 Test and performance evaluation

To validate this implementation a test was made to see if the system worked correctly and additionally evaluate this implementation performance based on factors mention in chapter two.. Because we have split the synchronization into operations performed at the synchronization server and the client device we can measure them separately.

In terms of the end device computational load, this factor depends on the computation effort to operate on the SQLite database. Based on SQLite the official documents, using an Athlon CPU running at 1.6GHz with 1GB or memory, an IDE disk drive, and RedHat Linux 7.2 operation system with a stock kernel, SQLite has better performance than MySQL. [32]. In this testing bed when using, a T2300 CPU running at 1.66GHz with 3GB memory and connecting to a hard disk with average writing speed of 35MB per second and an average reading speed of 37 MB per second, we did insert 10000 records in the SQLite (with transactions) in 0.452 second. In this test, 10000 records synchronization time takes 1.43 seconds. The size of the binary SQLite files containing these 10000 records was 48KB; therefore the end device computation load was very low. Note that this file size was based on new data of average size 4 KB arriving at a rate of 840 new records per minute and synchronization occurs every 1 minute.

In terms of the solution extension, because this synchronization systems design has similar with a version control system, it should be feasible to extend this design to support a nearly unlimited number of devices for synchronization with a central server. The actual performance with a given central server will of course depend upon the details of the number of clients their rates of record updates, the size of these records, etc. Based upon this test we expect that the system should scale roughly as that of a file server excluding the extra work required for the data updated and merges.

Regarding the error handling ability, because the data exchanges between source and target devices utilizes SFTP the underlying TCP connection and the application layer protocol features of SFTP will ensure data integrity during the transmission process [33]. Therefore this mechanism would have an acceptable error handling ability.

5 Conclusions and Future Work

This chapter concludes the thesis and outlines future research directions.

5.1 Conclusions

The overall focus of this thesis was to design a "repository" based SQLite database synchronization system that could operate in non-real-time. Throughout the thesis we have shown that synchronization brings clients device benefits in terms of short synchronization times, low bandwidth consumption and via the underlying database function a rollback function when used in conjunction with a repository based synchronization system.

From the end devices perspective, this design requires a minimal computation power on the end clients, because this solution was designed to be a non-real-time and it does not require continuous link layer connectivity between the client devices and the servers. Since this design was based on the underlying concepts of a version control repository, this synchronization system offers some of the benefits from version control system, for instance data rollback.

From the SQLite database perspective, since SQLite does not support multiple users at a time and it does not support functionality which is supported by most distributed database system, for example transaction log. Although SQLite has better performance in local applications than other database such as MySQL or PostgreSQL, by combining SQLite with a repository with a MySQL (or other) database it is will not only deliver a better performance for local application, but also support multiple-users, transaction logs and supporting roll back functionally. Such a combined SQLite and repository solution to could be used as distributed database system as well.

From the server perspective, in our test case, we ran on top of a Microsoft Windows® platform, but the same approach could be used to run on a Linux platform. Similarly the client was run on a Microsoft Windows platform in the test, but it could also be run on top of Linux, Apples Mac OS, or one of many embedded operating systems such as Android or iOS. In fact, the client only needs to support the SQLite API and the SFTP a TCP/IP stack.

5.2 Future work

Even though the results show that this repository based solution works well with SQLite, a number of assumptions concerning the testing platform and the rate at which data is generated, the local memory capacity and local processing capacity, and the times when synchronization occurs. To precisely quantify the performance in a real application of this system, additional testing in a real network environment and a large amount of user testing, needs to be performed. This leads us to our suggestions for future work. Additionally, this testing environment mentioned in the previous section a limited number of machines, we have not tested the performance of this implementation on a wide variety of hardware and operation system (OS) platforms. This part would remain for a future work.

Other than those, according to the chapter three pointed out, encoding difference records to SQLite database binary file would provide a lower file length compare to the

SQL statement textural file. This argument is based on the assumption that the SQL statement file without any compression. Therefore, it should be taken in to consideration that encoding the difference records to SQL statement textural files with properly compressed would even have lower file length than SQLite binary database format. Thus, more analysis work regarding different encoding and compression mechanism should be considered as a future work. Furthermore, the chapter four was lack of conflict recording handing solution analysis, this issue also should be done in the next step.

This thesis can be considered as an initial starting point for a repository database synchronization system. There are several aspects of the current design that could be optimized. The client authentication process is performed based upon exchanging a user name and password via the SFTP server; this could be replaced by the use of LDAP[14].

Finally, since SQLite databases are widely used in applications running locally, or in applications running on a handheld device, we need to examine the performance on a variety of devices and operating system platforms. In our test, both the client side application and server side application ran on one of the Microsoft Windows platforms. Therefore, an obvious next step is to run the client code on Microsoft's Windows Mobile or the Android, and to run the server on a Linux server platform. With regard to the server it would be interesting to explore executing the server on a virtualization server, such as Amazon's EC2 platform. Such a solution would enable a service to easily and rapidly expand to support a large number of users.

[14] Lightweight Directory Access Protocol (LDAP) is an application protocol for querying and modifying data of directory services implemented in Internet Protocol (IP) networks

References

1. **Shah, Agam.** Global computer usage, cell phone ownership jump. *InfoWorld.* [Online] InfoWorld, 10 05, 2007. [Cited: 07 01, 2010.] http://www.infoworld.com/t/hardware/global-computer-usage-cell-phone-ownership-jump-956.

2. **Unit's, Economist Intelligence.** *Global computer ownership will continue to rise in 2010.* Economist Intelligence Unit's. 2010. Web report. http://www.economist.com/node/15062710.

3. **Factbook, CIA World.** *Cell Phone Usage Worldwide, by Country.* CIA World Factbook 2009. 2009. Information Please® Database, ©.

4. **Agarwal, S., Starobinski, D. and Trachtenberg, A.** *On the scalability of data synchronization protocols for PDAs and mobile devices.* 4, 1984, Vol. 16, pp. 22-28. 10.1109/MNET.2002.1020232 .

5. —. **Agarwal, S., Starobinski, D. and Trachtenberg, A.** 4, Boston : IEEE Communications Society, 2002, Vol. 16. 0890-8044.

6. **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.** *Version Control with Subversion.* Stanford : TBA, 2008. 94305.

7. **CVS.** CVS--Concurrent Versions System v1.12.12.1: Overview. *ximbiot.com.* [Online] ximbiot.com. [Cited: July 25, 2010.] http://ximbiot.com/cvs/wiki/CVS--Concurrent%20Versions%20System%20v1.12.12.1:%20Overview.

8. **Tamilmani, Karthik.** *Studying and enhancing the BitTorrent protocol.*: Stony Brook University, 2006.

9. **Ericsson, IBM, Lotus, Matsushita Communications Industrial Co., Ltd., Motorola, Nokia, Openwave, Starfish Software, Symbian.** *SyncML Sync Protocol, version 1.1.* [Electronic Document] 2002. Version 1.1.

10. **Microsoft Corporation.** Exchange Server Protocols Document Roadmap. *Microsoft Corporation.* [Online] Microsoft Corporation, July 29, 2010. [Cited: July 31, 2010.] http://download.microsoft.com/download/5/D/D/5DD33FDF-91F5-496D-9884-0A0B0EE698BB/%5BMS-OXDOCO%5D.pdf.

11. **MySQL.** 10.3.1. The DATETIME, DATE, and TIMESTAMP Types. *MySQL 5.1 Reference Manual.* [Online] MySQL. [Cited: July 10, 2010.] http://dev.mysql.com/doc/refman/5.1/en/datetime.html.

12. **Ben Escoto,Andrew Ferguson.** rdiff-backup. *rdiff-backup.* [Online] rdiff-backup, Febrary 14, 2009. [Cited: October 01, 2010.] http://rdiff-backup.nongnu.org/.

13. **Ben Escoto, Kenneth Loafman.** duplicity. *duplicity.* [Online] duplicity, September 19, 2010 . [Cited: October 01, 2010.] http://duplicity.nongnu.org/.

14. *Building an Industry-Wide Mobile Data Synchronization Protocol.* **Paper, SyncML White.** Vol. 1.0.

15. **SyncML.** *SyncML Sync Protocol, version 1.1.* 2002.

16. **SyncML.** SyncML Representation Protocol, Data Synchronization Usage v1.1. *SyncML.org.* [Online] SyncML., Feb 15, 2002. [Cited: June 30, 2010.] http://www.openmobilealliance.org/tech/affiliates/syncml/syncml_dm_represent_v11_20020215.pdf.

17. **Dumbill, Edd.** SyncML Reference Toolkit Manual. *XML Watch: SyncML toolkits.* [Online] June 06, 2003. [Cited: June 23, 2010.] http://www.ibm.com/developerworks/xml/library/x-syncml3.html.

18. *A space-optimal wait-free real-time synchronization protocol.* **Cho, Hyeonjoong, Ravindran, Binoy and Jensen, E.D.** s.l. : IEEE, 2005. 17th Euromicro Conference. pp. 79- 88. 10.1109/ECRTS.2005.5.

19. **Teliasonera.** Teliasonera 4G. *Teliasonera.* [Online] Teliasonera. [Cited: July 24, 2010.] http://www.teliasonera.com/4g/.

20. **Slony-I.** *Introduction to Slony-I.* [Online] May 2011. http://slony.info/documentation/2.0/preface.html#INTRODUCTION.

21. **Marcotte, Ludovic.** Database Replication with Slony-I. *http://www.linuxjournal.com.* [Online] Slony-I, April 28, 2005. [Cited: July 24, 2010.] http://www.linuxjournal.com/article/7834?page=0,0.

22. **MySQL.** *Setting the Replication Master Configuration.* [Online] May 2011. http://dev.mysql.com/doc/refman/5.0/en/replication-howto-masterbaseconfig.html.

23. **Diehl, Mike.** Database Replication with Mysql. *linuxjournal.* [Online] linuxjournal, May 25, 2010. [Cited: June 25, 2010.] http://www.linuxjournal.com/content/database-replication-mysql.

24. SQLite Is A Zero-Configuration Database. *SQLite.* [Online] [Cited: 06 16, 2010.] http://www.sqlite.org/zeroconf.html.

25. **sqlite.org.** About SQLite. *SQLite.* [Online] SQLite.org. [Cited: June 23, 2010.] http://www.sqlite.org/about.html.

26. **Veen, Jeffrey.** Polar Heart Rate Monitors: Gimme my data! *veen.com.* [Online] veen.com, November 2, 2005. [Cited: July 20, 2010.] http://veen.com/jeff/archives/000810.html.

27. **Lossless.** Lossless Data Compression. *http://www.data-compression.com/.* [Online] data-compression.com. [Cited: July 25, 2010.] http://www.data-compression.com/lossless.shtml.

28. **David Salomon (with contributions by Giovanni Motta and David Bryant).** *Data Compression: The Complete Reference. 4th Edition.* 2006. 1846286026.

29. **lab49.com.** Adobe AIR: Synchronizing SQLite Databases With The Server. *lab49.com.* [Online] lab49.com, November 19, 2007. [Cited: June 21, 2010.] http://blog.lab49.com/archives/1569.

30. **Ziegler, Ann.** Auto-Generated Update SQL. *vb123.com KB.* [Online] [Cited: 05 15, 2011.] http://www.vb123.com/kb/index.html?200202_az_auto_updates_sql.htm.

31. Auto-Generated Update SQL. *vb123.com.* [Online] Febrary 2002. [Cited: July 23, 2010.] http://www.vb123.com/kb/index.html?200202_az_auto_updates_sql.htm.

32. **SQLite.org.** Database Speed Comparison. *sqlite.* [Online] sqlite.org. [Cited: May 20, 2010.] http://www.sqlite.org/speed.html.

33. **Process Software.** A Comparison of Secure File Transfer Mechanisms. *http://www.process.com/.* [Online] [Cited: July 24, 2010.] http://www.process.com/tcpip/sft.pdf.

34. **Raj, Gopalan Suresh.** A Detailed Comparison of CORBA, DCOM and Java/RMI. *Web Cornucopia.* [Online] [Cited: 02 10, 2010.] http://my.execpc.com/~gopalan/misc/compare.html.

35. **Liu, Yan, Babar, Muhammad Ali and Gorton, Ian.** *Middleware architecture evaluation for dependable self-managing systems.* Karlsruhe, Germany : Springer Verlag, Tiergartenstrasse 17, Heidelberg, D-69121, Germany, 2008. 03029743.

36. **wikipedia.** Constructive research. *Constructive research_wikipedia.* [Online] wikipedia. [Cited: 12 23, 2009.] http://en.wikipedia.org/wiki/Constructive_research.

37. **Susan Thomas, Jed Hartman, and Judith Radin.** Introduction to RPC Programming. [book auth.] Jed Hartman, and Judith Radin Susan Thomas. *IRIX Network Programming Guide.* s.l. : sgi.com, 2003.

38. **Nomor Research GmbH.** *Technology of High Speed Packet Access (HSPA).* s.l. : Nomor Research GmbH, 2006.

39. *Draft summary minutes, decisions and actions from 3GPP Organizational Partners Meeting.* **3GPP .** Tokyo : NTT DoCoMo's FOMA, 2001.

40. **Charles Lin,.** Ascii vs. Binary Files. Lecture notes for CMSC311: Computer Organization, Spring 2003, *Computer science department of University of Maryland. 12 March 2003.* [Online] University of Maryland. [Cited: July 20, 2010.] http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/BitOp/asciiBin.html.

41. **W3C.** Extensible Markup Language (XML) 1.0 (Fifth Edition). *w3.org.* [Online] w3.org, November 26, 2008. [Cited: June 24, 2010.] http://www.w3.org/TR/REC-xml/#dt-doctype.

42. **Motorola.** *Long Term Evolution (LTE): A Technical Overview.* s.l. : Motorola, 2010.

43. **apache.org.** Subversion Documentation. *Apache Subversion.* [Online] Apache, June 21, 2010. [Cited: July 23, 2010.] http://subversion.apache.org/docs/.

44. **Dodig-Crnkovic, Gordana.** Constructivist Research and Info-Computational Knowledge Generation. [Online] 2009. http://www.mrtc.mdh.se/index.php?choice=publications&id=1882.

45. **Ahmad, S.Z. and Qadir, M.A.** *Terminal Mobility Services in the Middleware Environment.* Amman, Jordan : s.n., 2007. IEEE/ACS International Conference on Computer Systems and Applications. pp. 332-335. 10.1109/AICCSA.2007.370902.

46. *A Study of Discovery Mechanisms for Peer-to-Peer Applications.* **M. Kelaskar, V. Matossian, P. Mehra, D. Paul, M. Parashar.** Washington : s.n., 2002. 2nd IEEE/ACM International Symposium. p. 444. 10.1109/CCGRID.2002.1017187.

Appendix

A. Useful tools used when developing application

Table 9: Useful tools used when developing application

Name	Description	Source
sqlite-manager	Extension for Firefox and other apps to manage any sqlite database	http://code.google.com/p/sqlite-manager/
FileZilla Server	SFTP server	http://filezilla-project.org/
Wireshark	Wireshark® is the world's most popular network protocol analyzer	http://www.wireshark.org/
MySQL-Front	"MySQL-Front MySQL GUI for database changes, data editing, SQL queries and more"	http://www.sql-front.com/

B. Raw information collection

During the literature study process, raw information collection and provides practical support argument. In this thesis literatures can be divided into three categories. They are academic literatures web material, and Wikipedia. The academic literature was found via various academic databases access to these databases was provided by KTH's library. These sources mainly were used to support my technical analysis motivation parts the solution and when writing the conclusions. The reset of literature was primarily used as background information –particularly to explain technical terms and to cite example of practical solutions.

As to the academic databases, Inspec (EV2), Computer Science Bibliographies, and Scopus reference databases were mainly used during the research. In order to provide easily understood definitions of the many technical terms related to data synchronization and cross-platform software Wikipedia was used. Additionally some web based material was used as references. These were found using Google or other web search engines. I also discussed my idea with other group members at the company where I was working and with professors in our department.

During the academic database search step, a number of keywords are validated in a thesaurus, provided by "Inspec (EV2)", in order to get "Inspec (EV2)"controlled terms and make the search results more precise. Next controlled terms and uncontrolled terms were use to search in the academic database in order to have both precise and extended references. Table 10 shows some the controlled and uncontrolled terms from (Inspec) used during the academic database search processes.

Table 10: Search terms

Controlled terms[15]	**Uncontrolled terms**
Synchronization	Synchronization
database machines	database
meta data	
relational databases	
file organization	File system
	cross-platform
	Common Object Request Broker Architecture

The literatures are chosen by relevant level and publication data. The reference rate is used to ensure the reference's validity. We were able to find two journal articles and one conference article.

[15] In our case, the controlled terms are verified by "Inspec".

C. Code

The code can be checkout from SVN server

https://longst.dydns.org:8443/svn/Doc/Code_server/

D. Sample Data

Table 4 SQL

CREATE TABLE "Culture" ("ItemsID" INTEGER PRIMARY KEY ,"ProductsName" TEXT,"Sellingstore " TEXT, "Price" FLOAT)

Table 5 SQL

CREATE TABLE "Culture" ("ItemsID" INTEGER PRIMARY KEY ,"ProductsName" TEXT,"Sellingstore " TEXT, "Price" FLOAT, "SyncFlags" INTEGER)

TRITA-ICT-EX-2011:88

www.kth.se