iDIBS: An Improved Distributed Backup System

Faruck Morcos[1], Thidapat Chantem[1], Philip Little[1], Tiago Gasiba[2] , and Douglas Thain[1]
[1]Department of Computer Science and Engineering, University of Notre Dame,
Notre Dame, IN 46556, United States
{amorcosg, tchantem, plittle1, dthain}@nd.edu
[2]Department of Electrical Engineering and Information Technology,
Technische Universität München, Munich, 80333, Germany
{tiago.gasiba}@gmail.com

Abstract

iDIBS is a peer-to-peer backup system which optimizes the Distributed Internet Backup System (DIBS). iDIBS offers increased reliability by enhancing the robustness of existing packet transmission mechanism. Reed-Solomon erasure codes are replaced with Luby Transform codes to improve computation speed and scalability of large files. Lists of peers are automatically stored onto nodes to reduce recovery time. To realize these optimizations, an acceptable amount of data overhead and an increase in network utilization are imposed on the iDIBS system. Through a variety of experiments, we demonstrate that iDIBS significantly outperforms DIBS in the areas of data computational complexity, backup reliability, and overall performance.

1. Introduction

Backing up important data has long been a standard practice. Clients traditionally rely on local storage devices to store data, but maintaining such a centralized, full backup system is expensive, especially when a large amount of data must be stored. In addition, the backup device (e.g., hard disks, tapes) is both an efficiency bottleneck due to its slow bandwidth and a single point of failure, unless some redundancy (which requires additional expensive hardware) is used.

Many computer systems have large amounts of unused disk space and long periods of inactivity. In a peer-to-peer arrangement, these systems can cooperatively store each other's data in their unused space. This largely eliminates the cost of backup hardware, reduces the space limitations, lessens the bandwidth bottleneck, and (depending on implementation) removes the single point of failure. At the same time, however, this kind of system introduces new concerns about reliability and recovery, security, performance, and fairness.

In a peer-to-peer backup system, reliability and recovery are major concerns due to the volatility of the networks. Peer nodes are free to enter and leave the network at will. If a client is storing some data on a peer that leaves the network, the client must decide whether to backup the data again or wait for the peer to reconnect. If a peer leaves after a client's disk has failed and before the client attempts to retrieve its data, recovery may not be possible unless some kind of redundancy (or error correction) is used. On the other hand, if a peer is storing some data for a client that leaves the network, the peer must decide whether to keep or discard the data. Another challenge is identifying peers with a client's data after the client has crashed. The list of a client's peer nodes could be stored on a permanent backup system, but this would sacrifice some of the benefits of a peer-to-peer approach, especially if the peer list changes frequently.

Another factor affecting performance is client data volatility. In the simplest implementation, when a file is modified, it is retransmitted to peers in its entirety, even if a similar version is already stored on those peers. A file which changes frequently can therefore generate a large amount of network traffic and occupy a large amount of storage. If error correcting codes are used for redundancy, as in many peer-to-peer backup systems, this can also involve a large amount of processor time. Similarly, to protect sensitive data from untrusted peers, data is typically encrypted, further increasing the computational expense of data volatility. Finally, to ensure fairness, some mechanism must be incorporated into the peer-to-peer backup system so that no peer can store its data without reciprocating.

We designed several solutions to address some of the previously mentioned challenges in a peer-to-peer backup system. To improve reliability, we modified existing mechanism for packet transmission. To facilitate recovery after a crash, we store information on each peer to help identify the other peers. Also, we implement an efficient erasure code to reduce computation time and thus enhance scalability of the system. We call this system iDIBS, to the best of our knowledge, it is the first to introduce the use of Luby Transform erasure codes in a peer-to-peer distributed backup system.

This paper focuses on the reliability, recovery, and performance factors of our solutions, since we identified these as potential areas of improvement in DIBS [16] (a *sourceforge* project of a peer-to-peer backup system). In the next section, DIBS and iDIBS are related to existing work in the area. Section 3 discusses the existing DIBS system. Section 4 describes the concepts underlying the changes implemented in iDIBS. Section 5 gives the details of the actual implementation of the changes and Section 6 describes results, as well as measurements of those changes. The paper concludes with a discussion of potential areas of improvement (Section 7) and a general assessment of iDIBS (Section 8).

2. Related work

There has been a considerable research effort in the area of peer-to-peer backup systems. Typically, the primary objective of these systems is to provide reliability and security without sacrificing the autonomy of each node. Pastiche [6] and Samsara [7] are two examples that share with DIBS the goal of decentralized peer-to-peer backup systems. In addition, [4] proposed a decentralized storage cluster architecture, CoStore, where nodes share equal responsibilities in storing data using network attached storage devices.

Usually, peer-to-peer backup systems replicate data redundantly onto multiple nodes. However, since nodes in a peer-to-peer network constitute an untrusted environment, the data is not simply copied; a significant body of research advocates data encryption (e.g. [1] and [9]), as well as authentication [8], in order to ensure privacy.

Another concern in an untrusted environment is freeloaders–nodes that store their data on the network without providing storage space in return. To address this problem, pStore [1], Pastiche [6], Samsara [7], and PeerStore [12] use symmetric trading, in which each node must provide the same amount of storage as it uses for its own backups. A more flexible variation of this idea is Storage Auctions [5], a bidding mechanism in which peers advertise their needed and available space, and pair up with complimentary nodes. This mitigates the threat of freeloading while allocating space where it is needed. Fairness can also be enforced using a decentralized, zero-sum trading system where each node is treated as an autonomous agent [10]. More recently, [11] proposed the use of game theory to reward nodes with good reputation and, at the same time, minimize freeloaders by giving nodes an incentive to share resources.

Most peer-to-peer backup systems attempt to optimize their storage methods for efficiency. Pastiche, for example, identifies data shared in common by several peers in order to minimize the quantity of data stored. The creators of pStore demonstrated that only a small number of replications are needed to ensure an acceptable level of redundancy, while the authors in [2] emphasized the need for small circle trading to ensure best efficiency. Moreover, network locality can be exploited for better performance [18].

3. Architecture of DIBS

The Distributed Internet Backup System (DIBS) was chosen as the platform for our implementation. DIBS is a readily available stand-alone software, which implements all of the common features of peer-to-peer backup systems that we wanted to improve, including the use of erasure codes for redundancy and the maintenance of an explicit peer list. DIBS also has other important features, such as asymmetric key encryption using GPG, which do not directly pertain to our modifications but are nonetheless significant characteristics of the system.

The basic idea of DIBS is to reach backup reliability through replications. Replication is needed in a peer-to-peer backup system, as the peer-to-peer network is usually quite volatile. In other words, peers may frequently come online/go offline. There are several methods to ensure robustness for the process of replication; one is through the use of erasure codes [13]. In DIBS, Reed-Solomon (RS) codes, explained below, are used to achieve this purpose. Another important design of DIBS is to have transparency under regular use. In other words, the user should not be bothered by the program during the normal operation of the system.

The RS codes are used to provide redundancy in DIBS by encoding the data and then splitting it into a number of blocks (or pieces). As long as a certain number of blocks are still accessible, the original data can be recovered. When DIBS backs up a file for a user, some number of blocks k is specified as the minimum number of blocks required for a successful recovery and is dependent on the file size. The ability to reconstruct a file with some minimum amount of data, irrespective of which pieces of data are missing, makes erasure codes suitable for use in a volatile, peer-to-peer network. In order to provide data redundancy, DIBS generates additional blocks. Altogether, the total number of blocks generated, transmitted, and stored among peers is $n = k+p$, where p is a user selected parameter that defaults to 2. Note that if the client is part of a small peer-to-peer network, it may be the case that some peers receive more than one block of the same encoded file. It is not difficult to see that such situation may lead to decrease in reliability, as more than one block of the file will be missing if one such peer goes offline.

An example of the DIBS backup process is presented in Figure 1, with $k = p = 3$ and $n = k + p = 6$. In this figure, a file is encoded and then split into different blocks, which are denoted by lettered squares. The blocks are then transmitted to peers for storage. The rightmost part of Figure 1 shows a possible scenario after the blocks have been stored onto `Peer1`, `Peer2`, and `Peer3`.

The redundancy level determines the maximum number of blocks that can be missing while still allowing the file to be recovered. For example, for the redundancy level of $\frac{n}{k} = \frac{2k}{k} = 2$, a file can be recovered if at least half of the peers, which store that file, are still online. Figure 2 illustrates this scenario with $n = 6$ and $k = 3$, showing that recovery is possible even when `Peer2`, which stores $\frac{1}{3}$ of the total number of blocks, is offline.

In order to provide security for the files, DIBS protects the data and file names using asymmetric key encryption. On the other hand, integrity is provided by hashing the files and calculating a MD5 value, which is a message digest, per file. The nodes keep a list of these MD5 values and use them whenever stored files are recovered.

The main operation of DIBS is as follows. The nodes found in a DIBS network may work as servers (S) or clients (C) or both. It is the responsibility of client C_1 to find $k + p$ servers when it wants to backup its data. Once $k + p$ servers $(S_1 \ldots S_{k+p})$ have been found, the data is stored and a list of these servers and the files are kept for future recovery. In the same way, S_i will keep a list of the blocks that it is storing, as well as the client to whom these blocks belong.

As nodes may come and go, there is a synchronization procedure to ensure that $k + p$ replicas of each file are stored over the network. In order to accomplish this, DIBS uses timeouts to determine whether a peer is still alive. A peer which has been unresponsive for more than d days (d is a user selected parameter that defaults to 10) is assumed to be permanently offline. During this time, data of the unresponsive peer that is stored on other peers remains intact. However, after the timeout period, a client will consider the peer to no longer be alive, remove that peer's data from its storage, and copy the data it had stored on that peer to a different peer. In this way, DIBS maintains a redundancy of p extra blocks in addition to the k pieces required for a successful recovery. Note that if more than p blocks become unavailable at the same time for a d-day period and if the owner of the file crashes, the file cannot be recovered since there is not enough encoded data to reconstruct the file. In the case where a peer that is believed to be permanently offline returns, its stored data is treated as invalid and should be deleted.

For a node to recover all backups (such as after a complete disk failure), it must be able to locate a sufficient number of peers that stored its data, as well as decrypt the data. This means that the peer list and GPG key for a client must be stored on some other types of backup system (such as removable media or dedicated backup hardware). If the latest backup of the peer list does not include at least k as many peers as originally stored a file, that file will be unrecoverable. The necessary frequency of peer list backups thus depends on the frequency with which nodes go offline. For a highly volatile network, this could require an unreasonable amount of user activity. The other way to recover all backups without a peer list is to exhaustively search for all servers that have the desired data. Clearly, this process could take a significant amount of time, and will substantially increase network traffic.

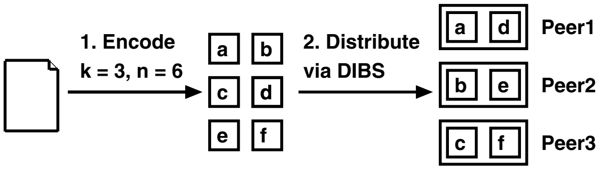

Figure 1. Encoding/splitting process in DIBS

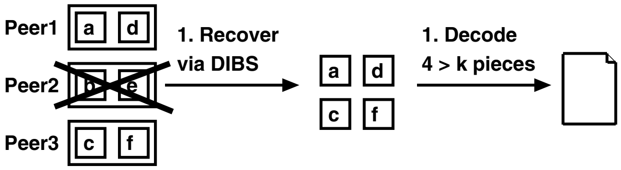

Figure 2. Decoding process in DIBS

4. iDIBS

Based on our analysis of DIBS, we identified three optimizations or improvements of this system, which we refer to collectively as iDIBS (improved DIBS). The three optimizations are: (i) implementing Luby Transform (LT) codes, described later, in place of the RS codes, (ii) improving reliability via modifications to packet transmission mechanism, and (iii) providing automatic peer list backups.

4.1. Erasure codes in distributed backup systems

Unlike uncoded data, encoded files provide improved reliability at the expense of data redundancy and encoding complexity [14]. The idea of using redundancy in backup systems is not new; the original design of RAID [17] uses a parity check to prevent failure in the case where a disk in the array fails. In 1960, the widely known Reed-Solomon

[19] (RS) codes were developed. These are non-binary block-codes that can be used as erasure codes and also as error-correcting codes. A block code produces n encoded symbols of length T bits from k source symbols, whereby $R = \frac{k}{n}$ is defined as the *code-rate*. RS codes have been widely used in many applications due to their remarkable characteristics, namely the fact that they are maximum distance separable (MDS) codes. This means that, if the code is used as an erasure code, whenever the decoder has available at least the number of source symbols k, then decoding will be successful. Such a property is highly desired in erasure codes. However, the penalty is that the encoding and decoding operations are very complex, i.e. $O(k^2)$. In practice only small source symbol sizes are used, e.g. 8 bits. Also, the number of encoded symbols in a RS code is determined by T.

The need for a code that has a performance comparable to MDS, is easily scalable and at the same time has small encoding and decoding complexities became more urgent as the data sizes to be transmitted became larger, especially in the Internet scenario. Although DIBS uses incremental backup to reduce the amount of data to be transmitted and stored, a general trend is that the size of new files increases as years pass by, thus emphasizing the need for more efficient encoding/decoding methods.

To improve broadcasting scenarios, Luby et al. introduced the concept of a *rateless code* [3] . These codes are also referred to as *digital fountain* (DF) codes. We decided to optimize DIBS by introducing the LT codes [15] into the system. LT codes are the first practical realization of a digital fountain and are based on a simple, scalable and irregular graph structure. In this code, the source symbols are encoded into a potentially endless number of randomly generated *encoded symbols*, i.e. $n \to \infty$, and therefore $R \to 0$. The code can easily manage large data lengths and file storage over multiple servers. Encoding and decoding complexity grows with the logarithm of the number of source symbols $O(\log(k))$, which means that these codes are less complex than RS codes. However, due to their random nature, these codes have a small reception overhead of about $k + O(\sqrt{k}ln^2(k/\delta))$ with a probability of recovering a file being $1 - \delta$, where δ is a configurable parameter of the code [15]. In practical terms this represents an $\epsilon = 10\% - 15\%$, which means that at least $(1+\epsilon) \times k = 1.1 \times k$ encoded symbols need to be available to the decoder (in average), for the decoding operation to succeed with high probability. Due to the random nature of the code, a so-called *encoded symbol identifier* (ESI) is required to identify each encoded symbol to the decoder so that it can setup the decoding graph properly. Decoding is usually done using message passing.

Other applications of LT codes include satellite communications file and video distribution over the Internet, and delivery of content to mobile clients in wireless networks [15]. We will demonstrate the advantage of using LT codes in lieu of RS codes in Section 6.

4.2. Packet transmission mechanism

We can improve the reliability of DIBS by modifying its packet transmission mechanism (when packets are being sent out to peers for backups). Consider a situation where a client wants to back up a file. DIBS responds to such a request by splitting the file into many different pieces and then storing the pieces remotely. The minimum number of pieces, k, needed for a full recovery is determined by the RS codes and is dependent on the size of the file under consideration. To cope with the situation where some peers may go offline, DIBS actually stores $n = k + p$ pieces. In other words, the client is provided with $\frac{k+p}{k}$ redundancy level.

When a peer that stores the client's file is considered to be permanently offline, DIBS will find different peer(s) to store additional pieces of that file in order to provide the promised redundancy level. Since DIBS stores p extra pieces of the file, one may assume that recovery is possible as long as up to p peers are offline. However, this is not always the case. Consider an example presented in Figure 3. The file to be stored is determined to have $k = 5$ and $n = k + 2 = 7$ (note that we are using the default value of p, which is 2). Furthermore, assume there are 5 peers in the network. In this example, the number of available peers needed for a successful recovery is not a constant; if any combination of two peers among `Peer3`, `Peer4`, and `Peer5` goes offline, the file is still recoverable and hence only three peers are needed. However, if either `Peer1` or `Peer2` goes offline, the total number of available pieces on the network reduces to 5 and thus, no more peers can go offline or the file is no longer recoverable.

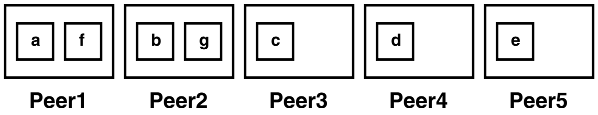

Figure 3. Packet transmission mech. (DIBS)

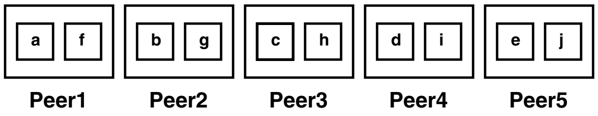

Figure 4. Packet transmission mech. (iDIBS)

As can be seen from the above example, DIBS cannot readily provide a recoverability guarantee based on the number of peers that can become offline. To remedy this problem, we modified the packet transmission mechanism as follows. We left the value of required pieces, k, unchanged. However, we developed a new formula to determine the number of pieces to be sent. Instead of storing $k + p$ pieces (as is the case in DIBS), we store only the required number of pieces plus some additional ones in such a way that if one peer is offline, the file can still be recovered (Figure 4). It is worth noting that if the network is large, our mechanism reduces to storing $k + 1$ pieces. The reason why we reduced the default data redundancy level of $\frac{k+2}{k}$ to $\frac{k+1}{k}$ is because the LT codes, discussed in the last subsection, introduce an additional overhead of about 15% of the original file size. In this way, we were able to address the weakness of DIBS, as discussed above, while keeping the amount of redundancy at a reasonable level. To increase the level of redundancy, the user may vary the value of p, which is defaulted to a small value of 2. However, it is in general unreasonable to expect the user to set the right value of p based on the characteristics of the network such as volatility, which change in time. For example, an optimal value of p given a file to backup depends on the number of required pieces as well as the number of available peers in the network at that instance. Requiring users to manually select p limits both the automation and adaptability of the system. To address this issue, iDIBS automatically selects the optimal value of p based on the actual network properties.

4.3. Peer list backups

Rather than requiring the user to manually backup the peer list every time it changes, we incorporate this backup process into the functionality of iDIBS. This reduces the frequency with which the user is required to perform some manual operation to keep the backups viable. It also brings the system closer to pure peer-to-peer backup, since less data must be stored offline. Perhaps most importantly, peer list backup increases the reliability of the entire system, since rather than relying on a (potentially outdated) backup of the peer list, every peer holds the most recent copy and only one is needed to recover the entire peer list.

Every time a peer is added or removed from the list, and every time the data stored on a peer changes, the modified peer list is backed up. Recording the addition and removal of peers is important for maintaining access to at least the minimum number of data blocks required for recovery. Recording which data is stored on each peer is also important for recoverability, since the system depends on that information to determine what files to recover (i.e., when a total recovery is being performed after a crash) and which peers store blocks for which files. It is important that this peer list backup occurs as soon as possible after a change, especially in more volatile networks.

Rather than distributing encoded blocks of the file as usual, an entire copy of the peer list is stored on each peer. Without this property, only a marginal reliability benefit would be gained by storing peer lists on peers. (In particular, the remaining benefits are that there is no single point of failure and that the peer list backups are kept current automatically.)

After a complete disk failure or crash, a user needs only rediscover a single peer in order to recover its peer list, and from there recover all stored data. Once this initial peer has been identified, no further searching is required and the system may proceed to the backup process. This is not a perfect solution, since the initial peer must still be identified, and (in order to decode data) the GPG key must be backed up as before. The former problem is simplified by the fact that the iDIBS approach guarantees that the stored version of the peer list is entirely current as of the time of the crash. The latter problem is likely unavoidable, but since the GPG key is a static block of information (unlike the peer list), it may be easily stored offline in any number of ways.

4.4 Design tradeoffs

Table 1 summarizes the expected advantages and disadvantages of iDIBS, all of which will be verified and further explained in Section 6. In general, iDIBS improves on reliability and scalability, while imposing data overhead.

Table 1. Tradeoffs of the iDIBS system

iDIBS	
Advantages	**Disadvantages**
LT codes are: - Faster than RS - More scalable than RS - Flexible transmit symbol size T, let us decide the performance of the encoder/decoder. (larger T, better performance)	- LT codes are probabilistic: they depend on δ to define a probability of decoding (nonetheless, for a properly designed system the probability of failure is negligible) - As T exceeds 256, decoding overhead becomes larger.
Increased reliability by improving packet transmission mechanism	- LT codes need a minimum of 15% of extra overhead plus ESIs to decode a file.
Peer Lists: - Allow for faster recovery - Increase reliability	- Peer Lists induce a small amount of overhead.

5. Implementation

In this section, we give the implementation details of our optimizations. The modifications regarding packet transmission mechanism, as well as those involving peer list backups, were implemented directly on top of the original DIBS in `Python` from the *Sourceforge* project. To incorporate the LT codes in lieu of the RS codes, we encapsulated a `C` implementation of the LT codes encoder/decoder modules, with the purpose of having an additional improvement in performance, via `Swig` (a `C/C++`wrapper), for use in iDIBS.

5.1. Luby transform codes

As part of our implementation, we developed encoder and decoder modules, which are designed to use the Digital Fountain concept of the LT codes. These modules are written in `C` but are later ported into `Python` for use in iDIBS.

The *LT encoder* reads a file of arbitrary size and calculates the redundancy needed for the encoded file. It then splits the file into pieces that are ready to be sent to peers in the network. An important parameter of the encoder is the symbol size T, which determines the number of encoded symbols given a file. Moreover, to construct the encoded pieces, the LT codes need a label called ESI (Encoded Symbol Id) to identify each symbol. Since ESI must be transmitted along with the encoded pieces, it will induce overhead to the backup process. Although not implemented in iDIBS, this extra overhead can be reduced by source coding (compressing) the list of ESI to be transmitted to each peer.

The *LT decoder* receives as parameters the encoded symbols, or a fraction of it if not all peers are alive, in addition to the ESI's. Other parameters for the decoder such as the *degree* of the distribution and the value of δ are predefined in the system. The only prerequisite for a successful decoding is to have over 115% of the original data with its corresponding ESI's. If a higher probability for successful recoveries is desired, the redundancy level can be increased to achieve this effect.

5.2. Packet transmission mechanism

This optimization was implemented by modifying parts of the DIBS source code that determine the total number of pieces to be transmitted over the network. Let us assume that the number of required pieces to recover, which is based on the file size, has been previously determined. In the original DIBS, the total number of pieces to be transmitted, n, is calculated as $n = k + p$. In iDIBS, recall that we want to determine the total number of pieces in such as way that if a peer becomes offline, the file can still be recovered. To accomplish this, we replaced the above calculation of n with a new code segment presented below in forms of pseudocode:

```
if:      $num > k$ then
         $n \leftarrow k + 1$
else:
         $n \leftarrow n * \lfloor \frac{k}{num-1} \rfloor$
end if
```

Here, num represents the number of live peers and k is the number of required pieces to recover. As can be seen from the above pseudocode, determining n is straightforward when the number of live peers num is greater than the number of required pieces k. In such situations, n can be set to $k + 1$. The more challenging situations happen when num is less than k. For example, assume client A has three live peers in its peer list and wants to store a file that requires at least four pieces to recover. If A only sends out $4 + 1 = 5$ pieces over the network, two peers $P1$ and $P2$ will receive two pieces and the third peer $P3$ will receive only one. In the case where $P3$ goes offline, the file can still be recovered since $P1$ and $P2$ collectively have four pieces. However, if either $P1$ or $P2$ goes offline, only three pieces can be obtained and thus the file cannot be recovered. To prevent this situation from happening, the above `else` statement ensures that recovery is possible as long as up to one peer is offline.

5.3. Peer list backups

To automate the backup of peer lists, the source for DIBS was modified at every point where a relevant change to the peers and backup data could occur. These include the places where a peer is added, where a peer is removed, where a peer discards the client's data, and where data is transferred to a peer. In each of these places, the internal representation of the peer list in DIBS changes. In iDIBS, this is immediately followed by writing out the internal peer list as a peer list file, and designating the file for automatic backup.

In order to ensure that an entire copy of the peer list is stored on each peer, the DIBS backup mechanism was augmented to check whether a file is the peer list, and to treat it as a special case. When the peer list is the file being stored, the erasure code parameters are set to treat the file as one data block, and to store it redundantly on every peer.

6. Results

The claims made in the previous sections are verified based on some experimental results presented below. All experiments were performed on 15 HP Workstations with 3.2 GHz Intel Pentium IV processor, 1 GB of RAM, 1 MB cache size, and Linux 2.4 kernel. Each workstation was connected to each other via LAN. The results shown in this section were obtained by testing the running version of iDIBS. For the case of erasure code comparison, the encoder/decoder modules were tested independently and offline in order to subject both the encoder and decoder to the same conditions and parameters. To assess the relative performance of iDIBS with respect to DIBS, representative file sizes (1KB-1MB) were used instead of complete backup system benchmarks. As the performance of the encoding/decoding mechanisms depends on the file sizes, and not the number of files being transmitted, experiments that are based on different file sizes are more appropriate for comparative purposes.

6.1. LT codes vs. RS codes

In order to assess our implementation of the LT encoder/decoder, we compare the performance of the RS implementation in DIBS against that of our LT implementation. Our objective is to prove that in practice LT codes are more scalable than RS codes and, as a consequence, more suitable for distributed backup systems. The performance benchmark is designed to test the encoders and decoders in terms of processing time, given several file sizes. We calculated the mean time of three encoding and decoding runs for files of the following sizes: 20KB, 40KB, 100KB, 250KB, 500KB, 750KB, and 1MB. In the case of our LT modules we performed two experiments with two different values of symbol size $T \in \{32, 256\}$. These values are used to illustrate the behavior of the encoder/decoder with medium size symbols 32 and large size symbols 256, although the symbol size may take any positive, non-zero value. The outcome of these experiments can be seen in Figure 5 through Figure 8.

We first tested the encoding time for both the RS and LT codes. As Figure 5 shows, the LT encoder outperforms the RS encoder with T of 32 for any size of the testfile. We can also see in Figure 6 that increasing T to 256 drastically improves the performance of the LT encoder. These two graphs clearly show the benefits and scalability of the LT codes. As for the decoding time, when $T = 32$, the RS codes outperform the LT codes for files larger than 200KB (Figure 7). However, when we increase T to 256, as shown in Figure 8, we observe a clear improvement in the decoding speed of the LT codes, which significantly outperform the RS codes.

Based on these results, it seems that as files to be stored get larger, the value of T should also increase for the use of the LT codes to be advantageous. Increasing the value of T comes at the price of increased redundancy; the network utilization naturally increases as the amount of redundant data grows.

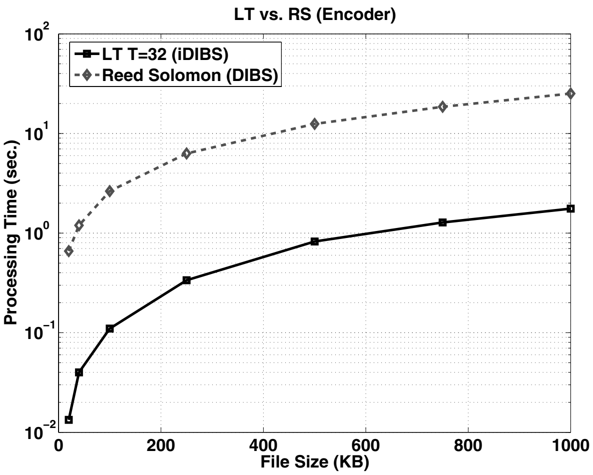

Figure 5. Encoder: processing time comparison for RS and LT, T=32

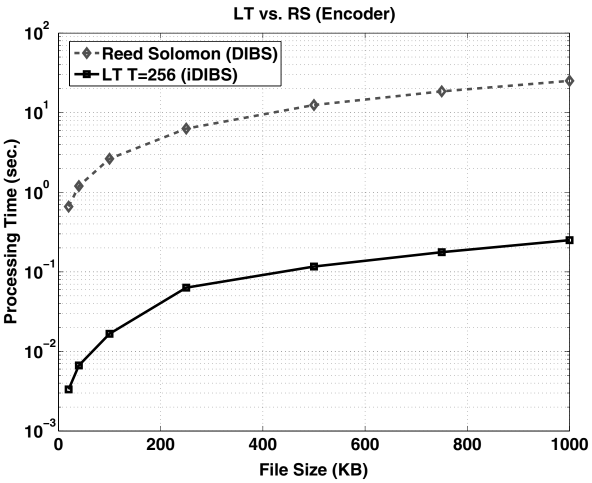

Figure 6. Encoder: processing time comparison for RS and LT, T=256

6.2. Packet transmission mechanism

To determine the difference in network usage between DIBS and iDIBS, we monitored the amount of data that was transmitted over the network for files of various sizes by first checking, for each file, the size of pieces that are ready to be sent out and then taking their sum. The test run was performed only once per file since the total packet size is constant given the parameters such as the file size and the total number of pieces to be sent.

Figure 9 illustrates the outcome of our experiments. The plot shows the network utilization per file, given its redundancy, in contrast to the original file size. In this scenario, $k = 5$ and the number of peers in the network is assumed to be greater than $k + 2$. The plot shows the performance of several systems; we have included a DIBS system with redundancy of 100%, i.e. $2k$, as a marker to compare with the original system using $k + 2$ and our implementation using $k + 1$ of redundancy. It can be seen that iDIBS with $k + 1$ and using RS codes has an expected reduction in network utilization. However, the final version of iDIBS using LT codes and the same redundancy philosophy of $k + 1$ has a slight overhead in network utilization per file due to improvement in scalability, performance and reliability.

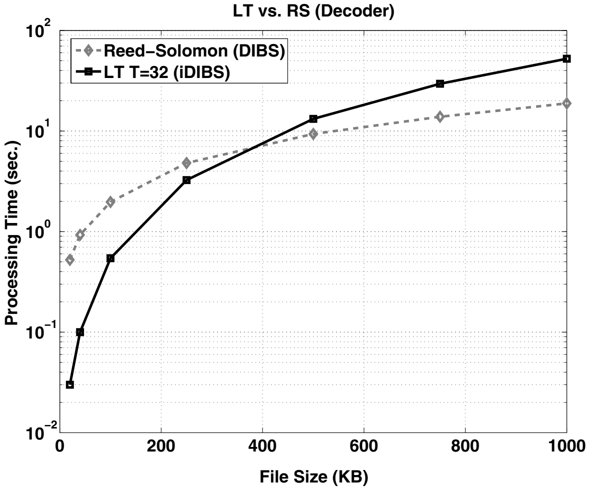

Figure 7. Decoder: processing time comparison for RS and LT, T=32

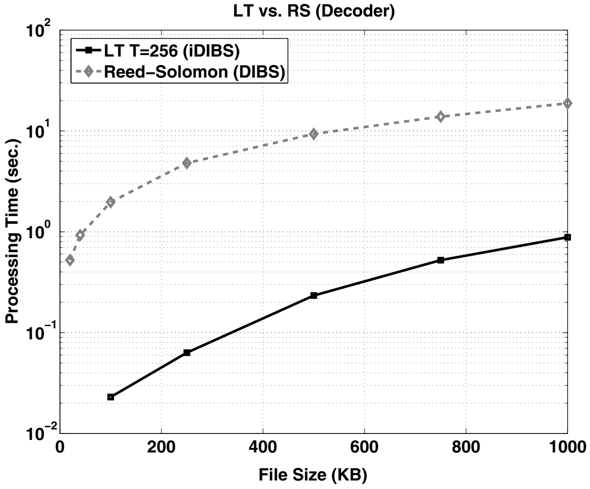

Figure 8. Decoder: processing time comparison for RS and LT, T=256

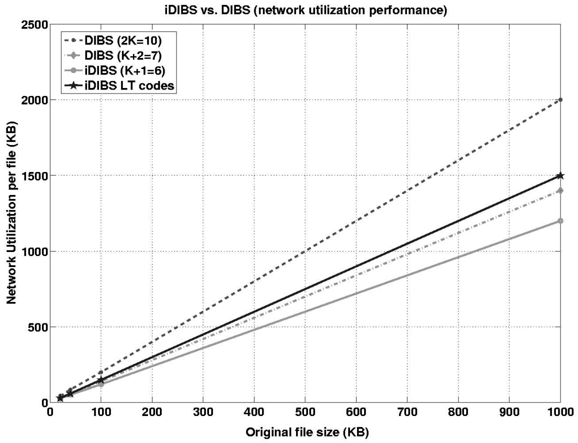

Figure 9. Difference in network utilization between DIBS and iDIBS

6.3. Peer list backups

The advantages of automated peer list backups can be evaluated in terms of correctness and efficiency. Correctness was verified by modifying the peer list in various ways, taking all but one peer offline, attempting a recovery, and checking that the returned file decoded to an accurate peer list. This demonstrated that the peer list is automatically and correctly stored for retrieval from any individual peer. We performed this experiment in various scenarios involving three or more network nodes.

To experimentally determine the amount of increased network traffic induced by the peer-list storage feature requires some assumptions about DIBS networks. The frequency of peer list modifications (and thus transmission) depends on network characteristics such as volatility. The efficiency of this feature can be analyzed numerically: A record for a single peer typically occupies less than 250 bytes in the peer list. A peer list with n peers, then, would occupy under $0.25n$ KB. Since this is distributed to every peer, the total transmission is $n^2 * 0.25$ KB. Defining network volatility as the probability p that a specific node will go offline (permanently) in a given period of time, the probability that at least one node goes offline (and thus the peer list changes) in that time period is $q = 1 - (1 - p)^n$. On the other hand, the average amount of data transmitted is $q * n^2 * 0.25$ KB. Considering that the average amount of data transmitted to maintain normal backup data is given by $p * d$ (where d is the amount of data backed up; the expected value of peers that go offline is $n * p$ and each peer holds d/n data), the cost of maintaining the peer list only exceeds the cost of maintaining the backup data when $n > \sqrt{4 * p * d/q}$ (where d is in kilobytes). For a client with only 50MB in backups on a network with $p = 0.5$

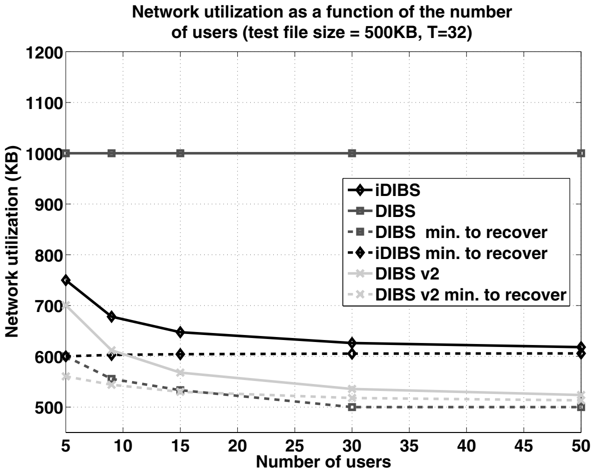

Figure 10. File size overhead in transmission as a function of the number of users

(being conservative), the value of n is 315 peers. The average list size could vary greatly with different sizes of DIBS networks; however, since most of the features of DIBS and iDIBS are designed for trusted networks with less than 50 nodes, it is safe to assume that in general usage, the cost of maintaining the peer list will be small relative to the cost of maintaining backups.

6.4. iDIBS tradeoff

Although we have demonstrated how we can improve DIBS through modification of several key architectural designs, whenever a system is exposed to fundamental changes, there exists a tradeoff in performance, reliability or complexity. This section is dedicated to quantifying the costs and drawbacks of iDIBS. We designed an experiment to illustrate the consequences of our modifications. This experiment involves the transmission of a file of fixed size, in this case 500KB, that is sent the first time a user performs a file backup. The idea is to quantify the network utilization of both iDIBS and DIBS as the number of users increases.

Figure 10 presents a series of curves showing the network utilization as the number of users increases. The solid lines represent the network utilization needed when backing up for the first time, while the dashed lines represent the minimum amount of transmitted data required to recover the files without any errors.

Again, a reference system with redundancy of 100% $(2k)$ is shown to reflect the tradeoff of network utilization present in the system when we aim to increase reliability. In addition, the plot shows the behavior of iDIBS, which certainly reduces the overhead of the backup system when compared with a system using $2k$ redundancy. On the other hand we can also see that iDIBS induces some overhead both in the initial data to be sent as backup and in the minimum amount of data required to correctly recover the backed up file when compared to DIBS using $k + 2$ redundancy. After looking at these results, a likely question would be regarding the tradeoff between DIBS and iDIBS. The overhead of iDIBS is mainly dependent on the inherent characteristics of the LT codes that need approximately a level of redundancy of the 15% of the file size plus the necessity to transmit encoded symbol identifiers (ESI), whose number is dependent on the transmit symbol size T. We can see how the dashed line of iDIBS has a flat behavior that represents the size of the file having the minimum needed for LT codes to decode, that is an extra 15% overhead, plus the size of transmitting the ESIs. This minimum is reached when we lose one peer. On the other hand, DIBS presents a curve allowing one extra piece to be lost, when the network size increases, and still can recover the backup file. This is a characteristic of the RS codes that allow to decode a file just by using an amount of data equivalent to the size of the file. The tradeoff is apparent: DIBS transmits less data (sacrificing performance and scalability) and theoretically allows the system to lose one extra user. Our design assumption implies that scalability, reduced computational complexity, and increased reliability are more important than the increase in network utilization introduced by LT codes and the reduction in redundancy level to $k + 1$. As stated earlier, we are convinced that the advantages of iDIBS offset its drawbacks and that, in overall, iDIBS outperforms DIBS.

7. Future work

One way this work could be continued is by integrating the components of iDIBS more tightly. Some components, though completed and tested individually, are not entirely integrated with the system as a whole. Complete integration would simplify use and allow large-scale and expected-use tests. This would also permit improvements to the existing Graphical User Interface (GUI), which has not yet been officially released at the time of this writing.

Testing is another area where more work is needed. It is difficult to know how suitable iDIBS is for different scenarios until it has been tested in them. Network volatility statistics would be particularly useful, along with information on the bandwidth, existing network load, and data volatility for common or expected uses.

Other useful extensions of this work would be the implementations of inter-peer communication when a node is going offline intentionally. Finally, to further improve computation speed and the decoding overhead inefficiency, the Raptor codes [20] can be implemented in place of the LT codes. The main advantages are that the encoding and decoding complexities grow linearly per source symbol and the decoding overhead is much smaller ($\epsilon \approx 0.1\%$).

8. Conclusion

By implementing an improved version of DIBS, we showed that our proposed optimizations are feasible and potentially beneficial to peer-to-peer backup systems in general. Our system maintained reliability by monitoring the network more intelligently. By also backing up peer lists, iDIBS not only improved recoverability but also stayed closer to the principle of peer-to-peer backup (though not quite achieving that ideal, since a user's GPG key must still be stored offline). The use of LT codes improved scalability in terms of the data encoding/decoding speed, making the system much more feasible for large backups, though at a cost in data transfer overhead. To the best of our knowledge, this is the first distributed backup system that uses Luby Transform codes, hence introducing a novel application to this revolutionary type of erasure codes.

In order to more rigorously verify the benefits of iDIBS, however, the system should be tested in a real user environment. for a longer period of time. While we have shown that iDIBS improves on DIBS in certain areas, it is not until these steps are taken that we can confirm iDIBS outperforms its predecessor. In the general case, this paper, nonetheless, provides evidence that it can happen in real user environments.

9. Acknowledgment

We would like to thank Emin Martinian for making his DIBS software available and for his invaluable comments and suggestions to an earlier draft of this paper. We thank our reviewers for helping us make the paper more accessible.

References

[1] C. Batten, K. Barr, A. Saraf, and S. Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[2] H. G.-M. Brian F. Cooper. Peer-to-peer data trading to preserve information. *ACM Transactions on Information Systems (TOIS)*, 20(2):133–170, April 2002.

[3] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review*, pages 56–67, 1998.

[4] Y. Chen, L. M. Ni, and M. Yang. Costore: A storage cluster architecture using network attached storage devices. *10th International Conference on Parallel and Distributed Systems (ICPADS 2004)*, pages 301–306, July 2002.

[5] B. F. Cooper and H. Garcia-Molina. Peer-to-peer data preservation through storage. *IEEE Transactions on Parallel and Distributed Systems*, 16(3):246–257, March 2005.

[6] L. Cox, C. Murray, and B. Noble. Pastiche: making backup cheap and easy. *ACM SIGOPS Operating Systems Review*, 36(SI), December 2002.

[7] L. Cox and B. D. Noble. Samsara: Honor among thieves in peer-to-peer storage. *In Proceedings of the 19th ACM Symposium on Operating Systems*, October 2003.

[8] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide area cooperative storage with cfs. *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, October 2001.

[9] P. Gauthier, B. Bershad, and S. Gribble. Dealing with Cheaters in Anonymous peer-to-peer Networks. Technical report, University of Washington, April 2004.

[10] B. Gelfand, A.-H. Esfahanian, and M. W. Mutka. An agent-based approach to enforcing fairness in peer-to-peer distributed file systems. *9th International Conference on Parallel and Distributed Systems (ICPADS 2002)*, pages 157–, December 2002.

[11] R. Gupta and A. K. Somani. Game theory as a tool to strategize as well as predict nodes behavior in peer-to-peer networks. *11th International Conference on Parallel and Distributed Systems (ICPADS 2005)*, pages 244–249, July 2005.

[12] M. Landers, Z. Han, and T. Kian-Lee. PeerStore: better performance by relaxing in peer-to-peer backup. *Proceedings. Fourth International Conference on Peer-to-Peer Computing*, pages 72–79, August 2004.

[13] M. Lillibridge, S. Elnikety, A. Birrel, M. Burrows, and M. Isard. A cooperative internet backup scheme. *Proceedings of the 2003 Usenix Annual Technical Conference*, pages 29–41, 2003.

[14] S. Lin and D. Costello. *Error Control Coding*. Prentice Hall, New Jersey, 2004.

[15] M. Luby. Lt codes. *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, November 2002.

[16] E. Martinian. DIBS: Distributed Internet Backup System. <http://sourceforge.net/projects/dibs>, 2004.

[17] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (raid). *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, September 1988.

[18] F. Picconi, J.-M. Busca, and P. Sens. Exploiting network locality in a decentralized read-write peer-to-peer file system. *10th International Conference on Parallel and Distributed Systems (ICPADS 2004)*, pages 289–296, July 2004.

[19] S. Reed and G. Solomon. Polynomial codes over certain finite fields. *SIAM Journal on Applied Mathematics*, June 1960.

[20] A. Shokrollahi. Raptor codes. Technical report, Laboratoire d'algorithmique, École Polytechnique Fédérale de Lausanne, 2003.