Carleton University
95.495 Honours Project

Peer-to-Peer File Sharing System using an Information Dispersal Algorithm

By Andrew Tytula, 237022
Supervisor: Professor Tony White

Abstract

Peer-to-Peer file sharing systems are more widespread today then ever. When the Internet was small, and computers slow, it was computationally infeasible as well as unnecessary to have peer-to-peer systems. As the Internet grew, server based systems are creating bandwidth bottlenecks that can only be overcame by a system that allows for direct peer-to-peer communication. With the closure of Napster, a server based Internet file sharing tool, other peer-to-peer file sharing systems have been made available.

While moving to a peer-to-peer solution allows for a service to be provided without the requirement of a server, the probability of a file sharing disruption is still more likely. A file can be stored on a peer and when the file needs to be retrieved, the peer may not be accessible (for many different possible reasons). Michael Rabin proposed and information dispersal algorithm where a file can be uploaded efficiently to many peers, where only a subset of these peers are required to rebuild the original file.

This document discusses the details of this algorithm as well as other algorithms that perform the same function. The document also discusses the implementation of this algorithm using a open source peer-to-peer protocol by Sun Microsystems known as the JXTA project.

Acknowledgements

In the preparation of this report, I have had assistance from many skilled individuals. First, I would like to thank Professor Tony White for his assistance and willingness to supervise me on this project. Without his inspiration, this report would not be discussing some of the more interesting aspects to file sharing such as Rabin's information dispersal algorithm. Secondly, I would like to thank Peter Goring, a colleague who allowed me to throw ideas off of him during the course of this project. Last, but not least, I would like to thank Professor Mike Just for giving me a perspective on the security issues that need to be considered in any file sharing system.

I would also like to thank Michael Rabin for discovering the IDA algorithm that makes this whole report possible.

Table Of Contents

1 INTRODUCTION 1

2 GENERAL INTRODUCTION OF P2P SYSTEMS 2
Client/Server Era 2
Introduction of P2P to End Users 2
The Rise and Fall of the Napster Empire 2
Different P2P Systems in Use Today 3
P2P Terminology 4
Security in P2P Systems 5

3 THE JXTA PROTOCOLS 5

4 INFORMATION DISPERSAL ALGORITHM 6

5 FILE SHARING DESIGN 9
System Description 10
Requirements and Assumptions 11
Actors and Components 13
Scenario Textural Descriptions 13
Message Sequence Charts 19
Test Strategy 25
The following test cases are to test the index files 25
Design Decisions 26
Security Concerns 28

6 CONCLUSIONS 30

7 REFERENCES 32

Table of Tables

Table 1: Guide to component identifiers. 10
Table 2: Summary of all assumptions. 12
Table 3: List of all functional and non-functional requirements 12
Table 4: A sample Scenario Textual Description is provided below. 14
Table 5: Table of STDs 15
Table 6: Table of MSCs 19

Table of Figures

Figure 1: Basic MSC notation (Boreleau, 1999) 19

1 Introduction

This report details the design of a peer-to-peer (P2P) file sharing system that works over any network, including the Internet. Although this sounds simple, the Internet being the largest network in the world, it is difficult design a true peer-to-peer application that does not rely on a server. I have chosen to work with a peer-to-peer tool known as JXTA, an Open Source project by Sun Microsystems. This tool will help simplify the design process and is introduced in detail in Chapter 3.

P2P file sharing systems are becoming popular these days due to the shutdown of server-based services such as Napster. A P2P system would allow for files to be shared without relying on a server, but is still subject to being censored based on content. P2P file sharing systems can be less reliable then server systems since most peers are personal computers which typically does not run 24 hours a day like a server. This is the motivation of the report, where I discuss a solution to this problem using the Information Dispersal Algorithm (IDA) invented by Michael Rabin in 1989. Rabin's algorithm provides a means of dispersing a file over many peers across the Internet such that only a small number of these peers are required to rebuild the file. It would be very difficult for all of the peers who contain the file to be censored and the probability of successfully recovering the file is high. In Chapter 4, Rabin's IDA is discussed in detail as well as another IDA.

Before I get into the details of JXTA and the IDA, I will give a brief introduction of P2P systems. This introduction will discuss the history of P2P systems, a brief overview of the different peer-to-peer systems in use today, and the definitions, which are needed when P2P systems are designed, implemented and used.

The conclusions of this report can be found in Chapter 6.

2 General Introduction of P2P systems

Client/Server Era

In the 1960s and 1970s, there was a network known as ARPANET (Advanced Research Project Agency NETwork). ARPANET was a network used by the military, which connected major military sites and universities in a national local area network (LAN). ARPANET consisted of machines, which provide a service and users/machines, which use a service. This formed client/server architecture. 30 years ago, P2P wasn't common due to computational and bandwidth limitations.

Since the number of network users in the world was still quite small, it was acceptable to use this client/server architecture. Today there are millions upon millions of people using the Internet, the biggest network on Earth. Unfortunately we are still dependent on a client/server architecture although there have been advances towards P2P systems.

Introduction of P2P to End Users

In October 1992, Microsoft introduces Windows for Workgroups, which integrated P2P networking into the Windows operating system. This allowed computers to network via a LAN and share resources such as files and printers without the need to buy and setup a server such as Microsoft Windows NT Server.

Even today, Windows supports P2P networking which is not that different from when it was introduced in 1992. The only draw back is that Windows P2P networking only works on an internal LAN and not over the Internet.

The Rise and Fall of the Napster Empire

In May 1999, a company known worldwide as Napster was founded. Napster created a software package, which allowed for users to share mp3 files across the Internet without having using a web browser. Before Napster was released, finding mp3 files was

difficult and for the layman, not easy. Napster made sharing mp3 files much easier, so easy in fact that the music publishers launched a lawsuit against Napster for copyright infringement due to loss of sales.

In the first half of 2001, Napster was ordered by the US courts to shut down its service. Although Napster is a P2P system, it still heavily relied on a Server to link everybody together. With Napster out of the way, other audio sharing applications were introduced, but they did not make the same mistake that Napster did. Most of these music-sharing applications do not heavily rely on a server. Some of these applications include Morpheus, AudioGalaxy and Gnutella.

Different P2P Systems in Use Today

As mentioned above, there are many P2P systems in use today, where a handful is being used for file sharing, specifically for audio files. The music pirating industry has been one of the major drives in P2P technologies. It should be noted that P2P file sharing has many more applications beyond its more common illegal purposes.

An emerging protocol, known as JXTA, sponsored by Sun Microsystems, is an open source project which provides developers with a standard for P2P computing without having to work from first principles. These protocols are discussed in more detail in chapter 3.

As the Internet grows, P2P systems are going to be used more and more. Client/Servers will not completely disappear since there are still obvious advantages such as web, email and DNS servers. Peers will no longer want or need to rely on a server to use a particular service. File sharing Systems such as Morpheus is just the start.

In businesses, an employee will use a P2P system to work from home or anywhere else in the world. Today we already see examples of this using VPN and software such as Symantec PC Anywhere.

At home, peers will use P2P systems to share files with friends, play computer games and even something as simple as watching television. To expand on the television example, we see that it would be easy for a peer to search for an advertisement for sci-fi shows and switch between the sci-fi "channels". In a non-P2P system, the peer would have to search for a television station, which broadcasts sci-fi shows, and then tune into that station. It is clear that the P2P solution can result in a quicker and more accurate search.

P2P Terminology

There are a handful of P2P protocols where each has their advantages and disadvantages, but they tend to all use similar terminology. The terminology that I will introduce is that used by project JXTA.

Every P2P system is made up of peers. A peer is a device, such as a personal computer, cell phone or PDA that interacts in some way with other peers. All peers must exist in a group, known as a peer group. A peer group is similar to a workgroup or domain in MS Windows where peers must be part of the same peer group in order to communicate. Typically, all peers are members of a main peer group. In JXTA this peer group is called NetPeerGroup.

Since there is no server that a peer registers with, there must be some way for peers to find other peers in which they can communicate with. A peer must advertise its presence and any service the peer offers. Other peers can search for the advertisements that contain information on how the two peers can communicate. Typically, peers communicate using a one-way communication pipe.

For the purposes of this report, the terminology introduced will be sufficient. Please refer to the JXTA protocol specification for further terminology.

Security in P2P Systems

User authentication is a path that is being explored by P2P Developers. User authentication traditionally relied on a server to maintain a list of user names and passwords but since P2P Systems don't typically rely on a server. An intuitive method involves the creator of a group admitting a peer and the peer must be authenticated with the group creator.

It is important that P2P systems ensure data security. Data security includes data integrity and data confidentiality. Since there are users who may want to ensure that the data is secure, it is important for P2P to at least work with the present data security products.

3 The JXTA Protocols

JXTA is a set of six protocols for P2P systems. P2P systems must implement at least one of these protocols to work with other JXTA systems. These protocols are

1. Peer Resolver Protocol (PRP)
2. Peer Discovery Protocol (PDP)
3. Peer Information Protocol (PIP)
4. Pipe Binding Protocol (PBP)
5. Peer Endpoint Protocol (PEP)
6. Rendezvous Protocol (RVP)

The PRP is used send queries to the handlers in the peer group and a facility to respond to the query. Handlers handle a query. There can be as little as zero handlers to as many handlers as are available.

The PDP is used to advertise and discover peer resources. A resource can be a peer, peer group, pipe, module, or any resource that has an advertisement. The default PDP is the one used for the "world group" a super group that all other groups spawn off of. If a peer group needs the PDP to do some specific function, the peer group can implement the modification for their peer group.

The PIP is used to gather information from a discovered peer. This protocol is optional and it is possible that a peer will not reveal any information.

The PBP is the way two peers can communicate. A communication pipe is similar to a water pipe. Information gets inserted at one end and is received at the other. Pipes are described using a pipe advertisement. There are currently three types of pipes: *UnicastType*, *UnicastSecureType*, and *PropagateType*.

The PEP is used to create a communication channel. The PBP uses this channel to communicate with other peers.

The RVP is used to propagate JXTA messages through a rendezvous peers. The means that a direct route is not required between two peers.

4 Information Dispersal Algorithm

The IDA is an algorithm introduced by Michael O. Rabin in his paper "Efficient dispersal of information for security, load balancing, and fault tolerance". The algorithm provides a way to break a file into n pieces such that only k pieces are required to reconstruct the file ($1 \leq k \leq n$).

Before discussing the IDA algorithm, I will describe a simple algorithm that has the same affect as M. Rabin's IDA algorithm, only the size of the pieces is larger and the pieces do not hide any attributes from the file. Although it is possible that others have defined this algorithm, I have found no evidence of this in my research. I will call this algorithm a Simple IDA algorithm (SIDA).

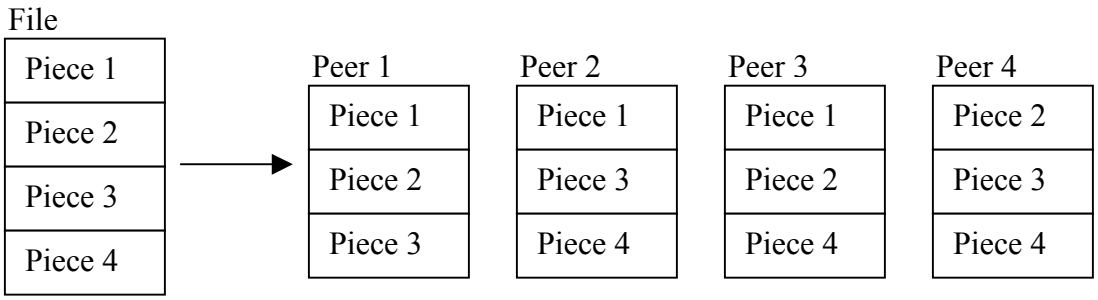
SIDA breaks the file into n pieces and distributes n-k+1 pieces to each of the n peers such that no peer gets the same n-k+1 combination of pieces. The file can be completely retrieved if all of the pieces can be retrieved by k of the peers. n-k+1 pieces

can be recovered from the first peer and each of the remaining k-1 peers provides an additional piece not yet recovered. The number of unique pieces recovered is

$$n-k+1+(k-1) = n$$

which is the n pieces of the file.

Example: Using the SIDA with n = 4 and k = 2



Getting the pieces from any two peers we can get all four pieces needed to rebuild the file.

The SIDA is a simple algorithm to understand and implement but it is not the most efficient algorithm. Rabin's IDA algorithm is accepted as the most efficient nk algorithm. It's key to success is that it interlaces the parity bits in each piece it transmits to each peer such that without the key, it would be impossible to determine if a bit is a parity bit or part of the file.

(Bestavros, 1990) provides most of the detail of Rabin's algorithm that will be regurgitated in this report. The idea Rabin conveys is that given a file $F = b_1b_2b_3b_4\ldots$ where each b_i in the stream can be viewed as an integer, to disperse the file a set of n linearly independent vectors (V_1, V_2, …, V_n) with dimensions 1xk must be generated. These vectors are known as keys. These keys are used to create the dispersal matrix A.

$$A = \begin{bmatrix} V_1^T \\ V_2^T \\ \vdots \\ V_n^T \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix}$$

Since A is an nxk matrix, it will take a kx1 matrix and map it to a 1xn matrix. By taking k integers at a time from F, and applying them to A, we get the integers to be sent to the n peers. The size of the file sent to each peer is |F|/k.

The reconstruction algorithm reconstructs the file k integers at a time. If there exists k sites (s_1, s_2, …, s_k) where the dispersed data can be obtained a matrix B is created consisting of the rows of the vectors represented by the site.

$$B = \begin{bmatrix} V_{s_1}^T \\ V_{s_2}^T \\ \vdots \\ V_{s_k}^T \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k1} & b_{k2} & \cdots & b_{kk} \end{bmatrix}$$

In theory, if we could obtain the dispersed info from all n peers, we could use A^{-1} to reconstruct F. The same philosophy can be used with the matrix B. We can use B^{-1} to find reconstruct F. Note that if less then k peers can be reached, the constructed matrix B could not be used to reconstruct F since B would be an axk matrix, where a is the number of pieces recovered, and we would multiply by a ax1 matrix which cannot be done. The matrix B^{-1} does exists because the keys were chosen such that they were linearly independent.

All arithmetic performed by the IDA must be carried out using the irreducible polynomial arithmetic (IPA). IPA views integers as polynomials over the field Z_p, where p is a prime number. (Bestavros, 1990) recommends choosing p = 2 which means integers are represented as polynomials with binary coefficients. If we only deal with 4-bit integers (0 – 15) they can be represented as:

0000 → 0 0110 → $x^2 + x$ 1100 → $x^3 + x$

0001 → 1	0111 → $x^2 + x + 1$	1101 → $x^3 + x^2 + 1$
0010 → x	1000 → x^3	1110 → $x^3 + x^2 + x$
0011 → x + 1	1001 → $x^3 + 1$	1111 → $x^3 + x^2 + x + 1$
0100 → x^2	1010 → $x^3 + x$	
0101 → $x^2 + 1$	1011 → $x^3 + x + 1$	

In this case, all arithmetic is performed modulo a reducible 4^{th} degree polynomial. (Bestavros, 1990) uses $(x^4 + x + 1)$. Since any 4^{th} degree irreducible polynomial will due, we will use $(x^4 + x + 1)$ as well. In order to add or multiply two integers, the integers must be converted into the polynomial form as listed above. These polynomials are added or multiplied modulo $(x^4 + x + 1)$ and the result is converted back to an integer. This can be done easily and efficiently using bit wise XOR and left non-circular bit shifts. Please refer to (Bestavros, 1990) for the algorithms to do this.

Analysis of the Two Algorithms

The SIDA is a simpler algorithm to implement and understand but is not as space efficient. In the SIDA, the file is broken up into n pieces, where each pieces size is |F|\n. Each peer gets n-k+1 pieces of the file and there are n peers so the space complexity is O(|F|(n – k)). Rabin's IDA algorithm stores |F|\k on each peer. Since there are n peers, the space complexity is O(n|F|\k) which is clearly more efficient then SIDA (since $k \leq n$). It is clear that uploading the whole file to the n peers (which is the same as choosing k = 1) would result in O(n|F|) for both algorithms. It can also be noted that since less information is being stored at each peer, then the network traffic will also be reduced.

5 File Sharing Design

This section uses Object Oriented (OO) Engineering practices to design the proposed file sharing system. It is important for any system to have a good design. With a good design, a person new to the system can easily learn how the system works and when a change to the system occurs. It is easy to propagate the changes through the design and into the implementation.

Typically, a good design process involves designing a system in iterations, where an iteration will make a relatively small advancement in the system. For this to be done with minimal chance of error, we need to have some method tracing changes throughout the design. Professor Bordeleau introduces the concept of traceability in his PHD thesis. The design for this file sharing system will use traceability as indicated in Professor Bordeleau's thesis.

Table 1: Guide to component identifiers.

ID	Category	Description
I: Introduction F: Functional requirement NF: Nonfunctional requirement A: Assumption S: Scenario Textual Description MSC: Message Sequence Chart DD: Design Decisions SC: Security Concerns		
I:PX-Y	System Description	Refers to sentence Y of paragraph X of system description
A-001	Assumption	00-99: all assumptions made during the course of this project
(N)F-001	CUB System	00-99: all functional and nonfunctional requirements related to the CUB
S-001	Scenario Textual Description	00-99 Scenario Textual Description following recommended template. (Bordeleau, 1999)
MSC-001	Message Sequence Chart	00-99: Message Sequence Chart following MSC'96 format generated from Use Case Maps.

System Description

The system is a file sharing system where users can upload a file to their peers in such a way that the file can be retrieved even if a number of peers are not accessible. Uploading the whole file to all of the peers can obviously do this but will waste space and will increase upload time. We need to reduce the size of the data sent to each peer but keep some redundancy such that the file can still be retrieved given a number of peers are inaccessible. There are algorithms that will break data into n chunks such that only k chunks are required to recover the data.

Every peer who wishes to upload files to other peers must also make space available on his computer. This is important since if nobody makes space available, then no files can be shared. A peer must advertise his space as available such that other peers can use the space.

When a peer wants to upload a file, first a list of peers who have advertised space must be obtained. From this list, we can choose a value for n such that $n \leq$ the number of the peers in the list and we can choose a value for k such that $1 \leq k \leq n$. With these choices we can create n chunks to be uploaded to n of the peers. When a file is uploaded to a set of peers, an index file must be created to keep track of where the files were uploaded.

When a peer wants to retrieve a previously uploaded file the peer must determine which peers contain which chunk and retrieve k chunks. This information can be obtained from the index file created when the file was uploaded. The k chunks can be used to rebuild the original file.

A file can be deleted by deleting all chunks stored at the peers. If k chunks are not deleted, the file can be rebuilt and hence would not be deleted. In theory, only $n - k + 1$ chunks need to be deleted to ensure the file cannot be rebuilt, but there would be wasted space.

This system is to be implemented using a P2P protocol. For simplicity and for ease of distributing the system, the P2P protocol which will be used will be JXTA, an open source P2P protocol managed by SUN Microsystems. A tool called the JXTA shell is part of the JXTA project and will be used to test the system.

Requirements and Assumptions

From the system description, a set of requirements can be generated. Requirements can be broken into two subclasses, Functional and Non-Functional.

Functional requirements usually describe a function of the system that will be present in the implementation. Non-functional requirements usually describe constraints in the system. Included with the requirements are assumptions made during design of the system. These requirements and assumptions will be traced through the whole design and into the implementation.

Table 2: Summary of all assumptions.

ID	Description	Traceability
A-001	The index file will not get corrupted	
A-002	When the user registers space, if the directory already exists, it was used for file sharing, and hence can be loaded.	

Table 3: List of all functional and non-functional requirements

ID	Description	Traceability
F-001	A file must be able to be uploaded to a peer.	
F-002	A file that was previously uploaded must be downloadable.	F-001
F-003	If the some of the n peers are not accessible but at least k of them are, the file can still be downloaded.	F-002
NF-004	The file must be broken down into smaller chunks and stored on peers in the smaller form.	
NF-005	A user must make space available to be used for file sharing before the user can upload files.	F-001
F-006	The user must advertise the amount of space that is available for file sharing and create an index file that.	NF-005, F-016
F-007	A user must obtain a list of peers who have advertised space available for file sharing.	NF-005, F-006
F-008	Given a list of peers, a user can upload a file to n of the peers.	F-001, F-007
F-009	The user keeps an index file for each file that is uploaded which instructs the user how to download and rebuild the file.	A-001, F-002, F-003
F-010	When the user wants to download or delete a file, the index file must be consulted to determine what peers contain what parts of the file.	A-001, F-002, F-009, F-015
NF-011	A file is considered deleted if n-k+1 pieces of the file are deleted.	F-015
NF-012	The system will be implemented using JXTA protocol.	

ID	Description	Traceability
F-013	If the index file cannot be read or written to, then the Server must abort execution.	NF-005, F-009, F-010
F-014	If the user requests more space then is available the Server shouldn't start	NF-005
F-015	A user must be able to delete a file	NF-011
F-016	If the index file does not exist, one should be created	

Actors and Components

Before moving to the next step in the OO Engineering process, the actors and components of the system should be defined. An actor is a user of the system and hence is typically external. An actor can be a person or another system. A component is a piece of the system that performs a set of tasks.

Actor Name	Actor Description
User	A person who uploads/downloads files and provides space for file sharing
Discovery Service	JXTA provides a discovery service used to discover peers. All advertisements must be published to this service.

Component Name	Actor Description
Init	The Init component initializes space for file sharing and starts the Server and Client components.
Server	The Server component stores files in initialized space and handles requests for uploading, downloading and deleting files.
Client	The Client component proxies the user by making upload, download and deletion of a file
Index	The Index component is used to read and write index files.

Scenario Textural Descriptions

This section outlines the Scenario Textural Descriptions (STD) for the file sharing system. The STDs describe the file sharing system scenarios in a structured textural format to organize the requirements.

Each scenario is organizes the requirements into a series of responsibilities starting with the triggering event and ending with the resulting event. A traceability column is added to the STDs to allow for backward traceability to the requirements of the system as well as traceability to other STDs (Bordeleau, 1999).

Each STD is defined in tabular form in terms of the elements listed below (Bordeleau, 1999). Each responsibility in an alternative scenario is assigned the same number as the responsibility it replaces.

- a unique identifier
- a brief textual description of the overall objective of the scenario
- the set of external actors that participate in the scenario
- a set of possible triggering events
- a precondition that must be satisfied in order to enable the execution of the scenario
- a sequence of responsibilities
- a post-condition that must evaluate to true after the execution of the scenario
- a set of possible resulting events
- a set of alternative scenarios
- a comment section that may be used by designers as a free format text window to specify different issues related to the scenario

Table 4: A sample Scenario Textual Description is provided below.

S-XXX: ***Scenario Textual Description title***	**Traceability**
Description: A brief textual description is entered here.	
External Actors: External actors to the system who participate in the scenario are entered here.	
Precondition: The precondition that must be satisfied allowing for the execution of the scenario is entered here.	
Triggering event: The triggering event is entered here.	
1. This is where the sequence of responsibilities is to be entered. 2.	
Postcondition: The postcondition that is evaluated to true is entered here.	
Resulting event: The resulting event is entered here.	
Alternatives: Alternative scenarios are entered here if needed.	
Nonfunctional requirements: Nonfunctional requirements that apply to the scenario are entered here if needed.	
Comments: Comments can be added here is needed describing the scenario.	

Table 5: Table of STDs

STD Identifier	**STD Title**	**Traceability**
S-001	Advertise space for file sharing	NF-005, F-006
S-002	Uploading a file	F-001
S-003	Downloading a file	F-002
S-004	Deleting a file	F-015
S-005	Starting the index component	F-009, F-010

S-001: Advertise space for file sharing	Traceability
Description: Prepare disk space for file storage and advertise the space as available for files.	NF-005, F-006
Actors: User, Discovery Service	
Precondition: No space has previously been made available for file storage nor has it been advertised	
Triggering event: User requests space to be made available	
1. User requests space to be made available	NF-005
2. Init creates the file storage directory and creates an starts the Index	S-005, NF-005, F-009
3. A module advertisement is published to the discovery service	F-006
4. A module specification advertisement is published to the directory service	F-006
5. An input pipe is created to receive commands from peers	
5.1. A pipe advertisement is published to the directory service	F-006
Postcondition: Server service is waiting for input via the input pipe	
Resulting event: Inform user that the storage space has been allocated and the Server service has started	
Alternatives: 2 – There is not enough space available, abort	F-014
2 – The server service does not have read/write access to the storage directory, abort	F-013
3, 4, 5.1 – The directory service is unreachable, abort	NF-012
Nonfunctional requirements: ▪	
Comments:	
▪	

S-002: Uploading a file	Traceability
Description: Steps taken to upload a file	F-001
Actors: User, Directory Service, n peers	
Precondition: The user has allocated space for file storage	NF-005
Triggering event: User makes an upload request	F-001
1. User makes an upload request providing values for the nk algorithm	F-001, F-003
2. Client obtains a list of peers with advertised space	F-007
3. n peers are chosen from this list based on performance	F-007
4. The nk algorithm is used to divide the file into n pieces	DD-001, F-003, NF-004
5. Each piece is uploaded the n peers	F-001
6. Provide the Index component of the specifics of the upload	F-009
Postcondition: The file has been uploaded and indexed	F-001, F-009
Resulting event: Inform user that the file is uploaded	
Alternatives: 2 – There are less peers with advertised space then the value chosen for n, abort	
6 – The index file could not be written, do not upload file	F-013
Nonfunctional requirements: ▪	
Comments: ▪	

S-003: Downloading a file	Traceability
Description: Steps taken to download a file	F-002
Actors: User, n peers	
Precondition: The file to be downloaded has an index file	F-010
Triggering event: User makes a download request	F-002
1. User makes a download request	F-002
2. Client obtains the n list of peers from the index file	F-010
3. Client retrieves pieces of the file from k of the peers (preferably from the fastest k peers)	DD-001, F-003
4. Using the nk algorithm, the file is restored from the k peers	DD-001, F-003
Postcondition: The file has been retrieved	
Resulting event: Inform user that the file has been retrieved	
Alternatives: 2 – The index file cannot be accessed, abort	F-013
3 – Less then k peers are accessible and hence the file cannot be retrieved, abort	F-003
Nonfunctional requirements: ▪	
Comments:	

▪	

S-004: Deleting the file	Traceability
Description: Steps taken to delete a file	F-015
Actors: User, n peers	
Precondition: The file to be deleted has an index file	F-010
Triggering event: User makes a delete request	F-015
1. User makes a delete request	F-015
2. Client obtains the n list of peers from the index file	F-010
3. Inform the Index component of the deletion	F-009
4. Client informs the n peers that the file is no longer required to be stored	NF-011
Postcondition: The file has been deleted	
Resulting event: Inform user that the file has been deleted	
Alternatives: 2 – The index file cannot be accessed, abort	F-013
4 – At most n-k peers are accessible and hence the file cannot be properly deleted. Delete the pieces from the accessible peers. Inform user that the file was not successfully deleted. Update the index file.	NF-011
4 – More then n-k peers are accessible but not all peers. Delete the pieces from the accessible peers. Inform user that the file has been deleted but pieces still remain. Delete index entry	NF-011
Nonfunctional requirements: ▪	
Comments:	
▪	

S-005: Starting the Index Component	Traceability
Description: Steps taken to start the Index component	F-009, F-010
Actors: None	
Precondition: The file sharing directory has been initialized	
Triggering event: Init starts Index	
1. Init starts Index	
2. Index reads the index file from the shared file directory	F-006
Postcondition: Index has been started	
Resulting event: Index has been started	
Alternatives: 2 – The index file could not be found, create a new empty index file	F-016
Nonfunctional requirements: ▪	
Comments:	
▪	

Message Sequence Charts

Message Sequence Charts provide a pictorial view of how the system runs through a series of end-to-end messages exchanged amongst the components of the system. The standard accepted today is MSC'96 which is the standard which will be used in this report. For more information on MSC'96, refer to (Bordeleau, 1999). Since this system is fairly simple in design, at this stage basic MSCs will be used. An example of a basic MSC is shown below.

Figure 1: Basic MSC notation (Boreleau, 1999)

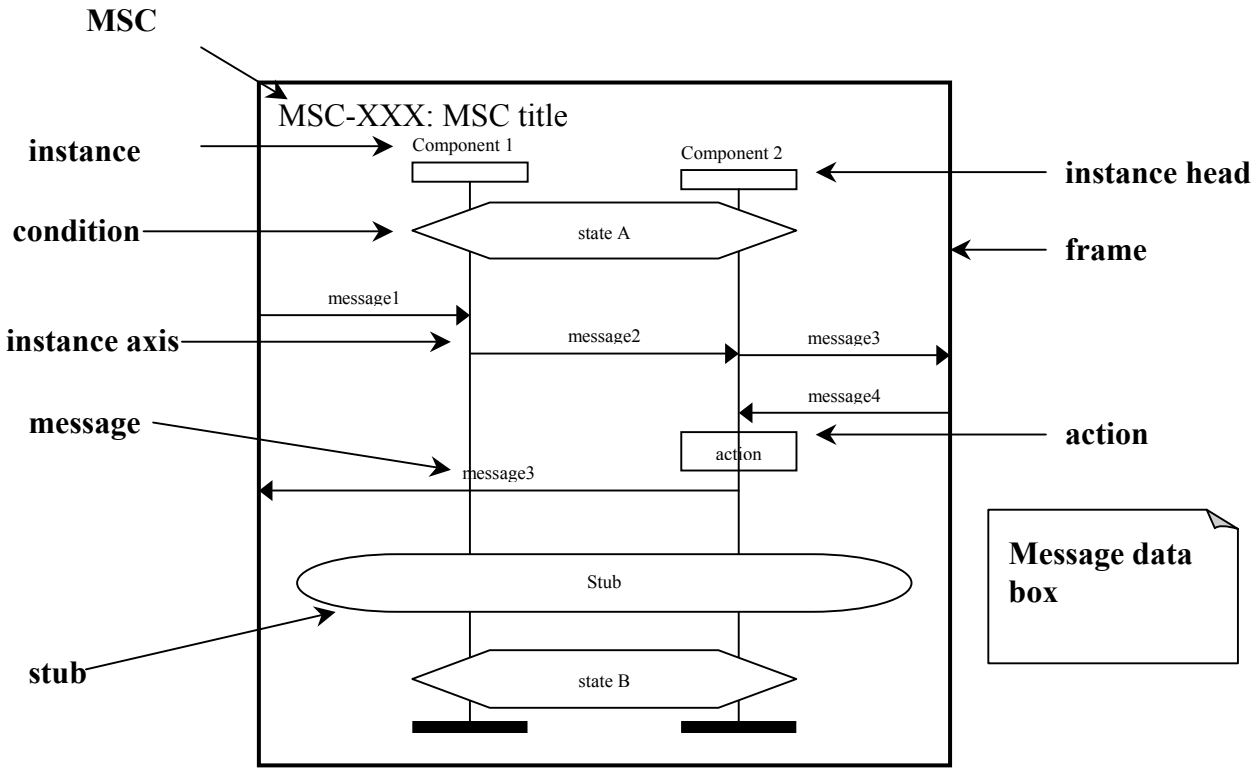
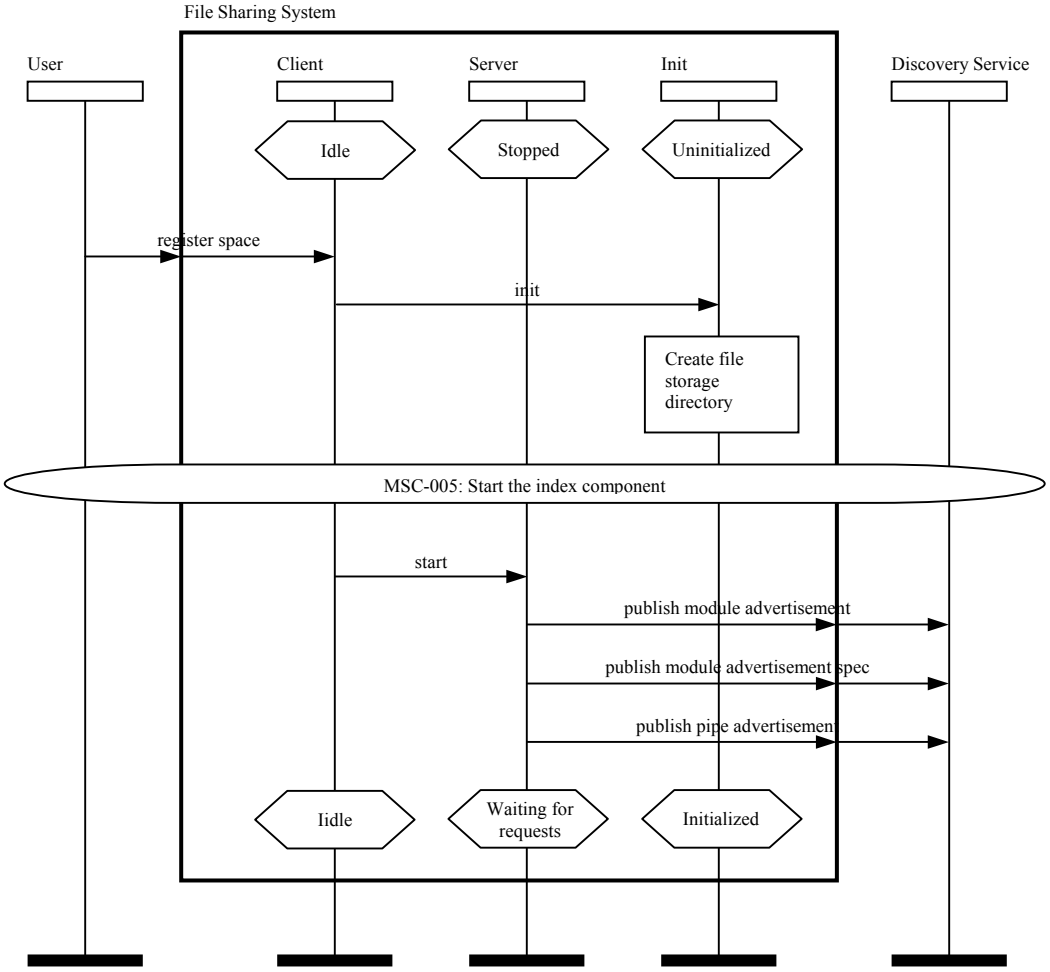


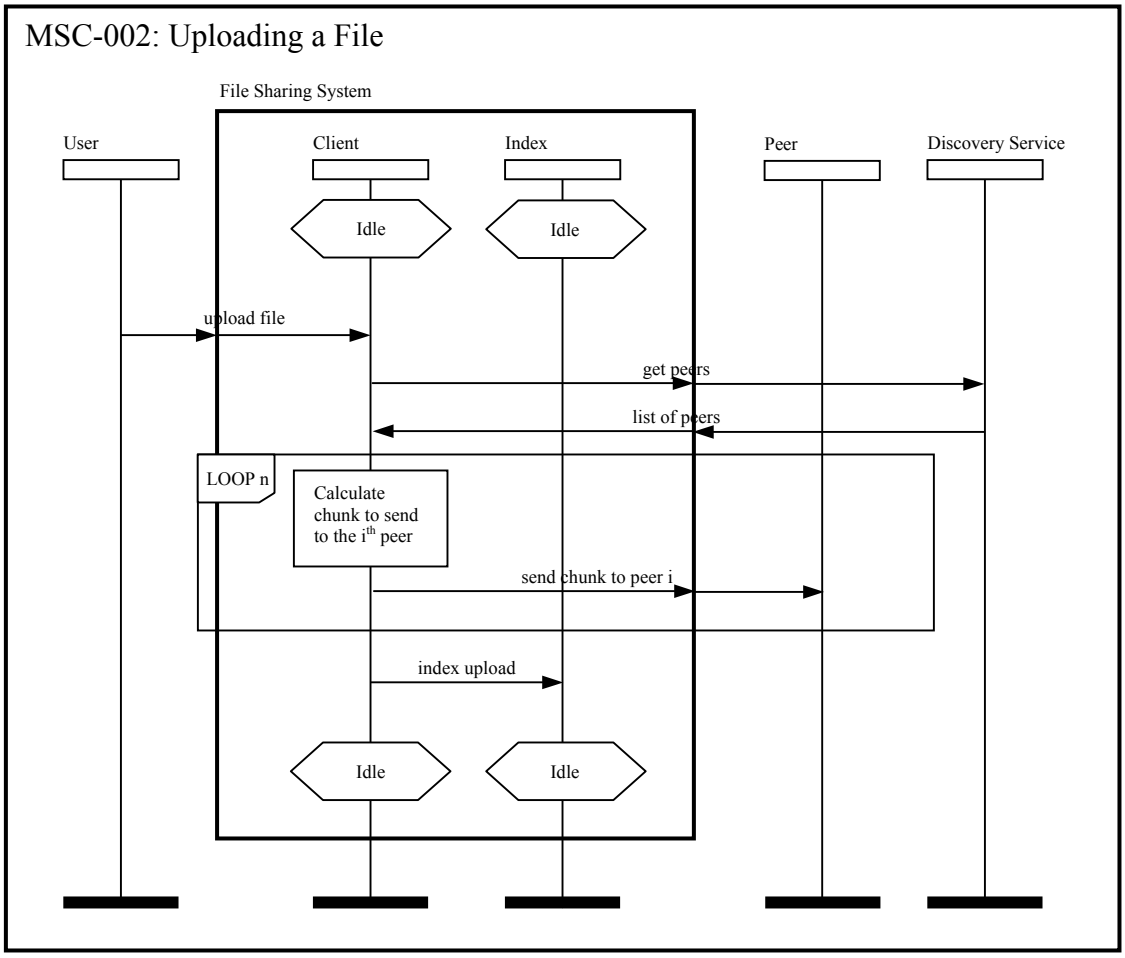
Table 6: Table of MSCs

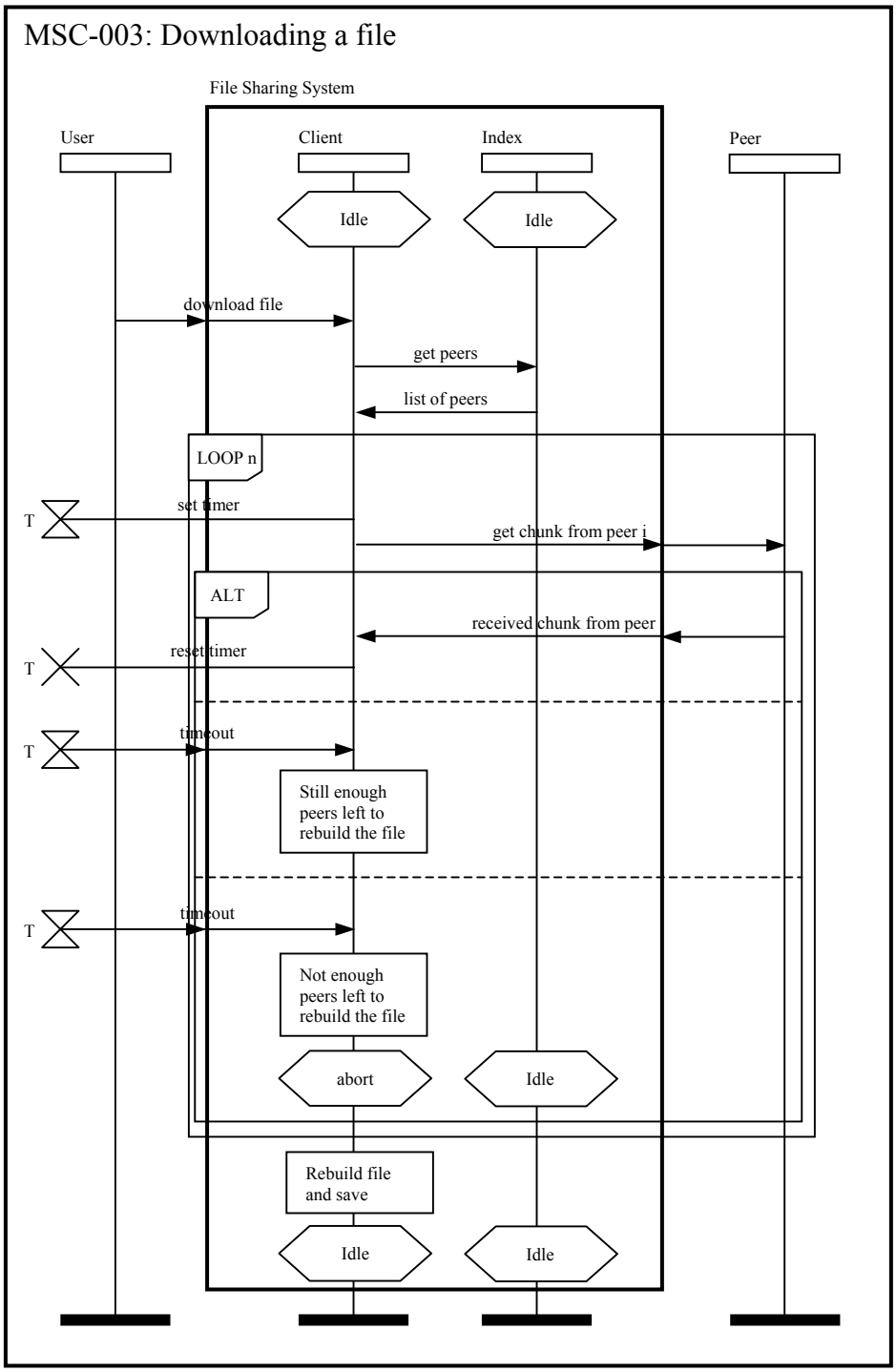
MSC Identifier	MSC Title	Traceability
MSC-001	Advertise space for file sharing	S-001
MSC-002	Uploading a file	S-002
MSC-003	Downloading a file	S-003
MSC-004	Deleting a file	S-004
MSC-005	Starting the Index component	S-005

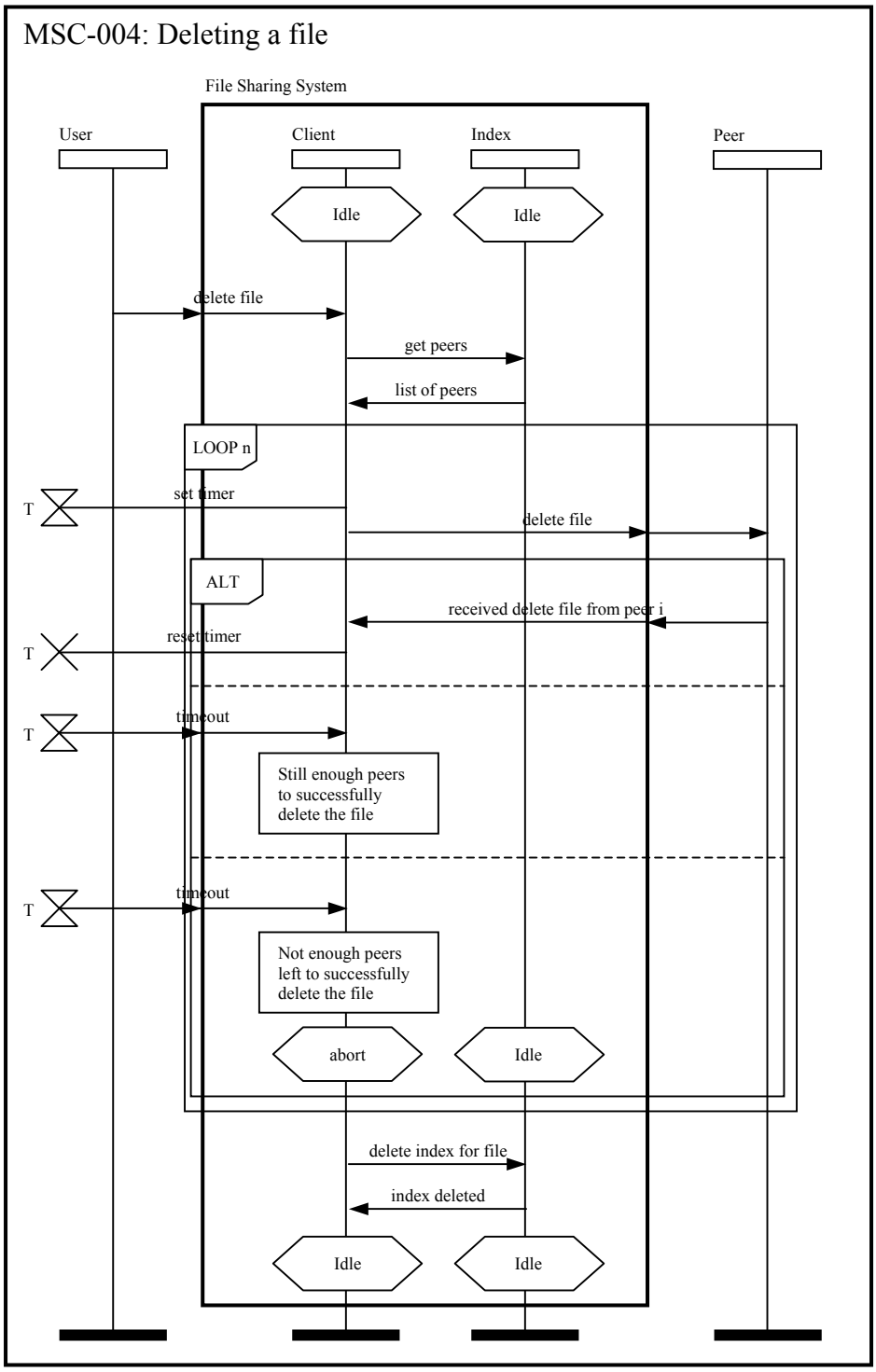
MSC-001: Advertising Space for File Sharing

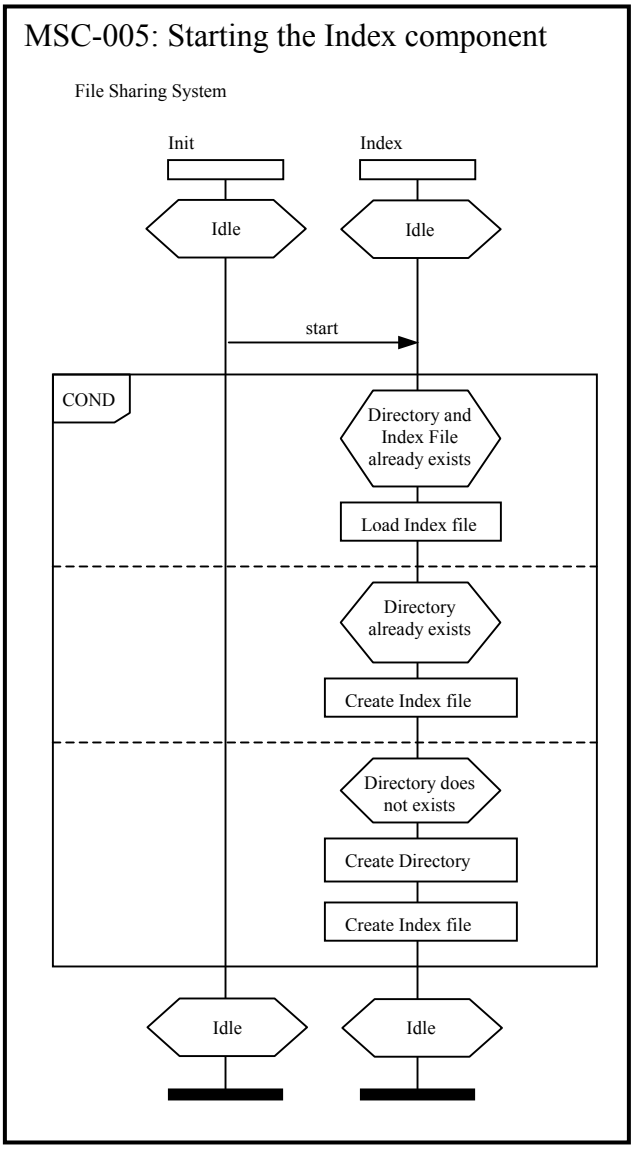


MSC-002: Uploading a File




MSC-003: Downloading a file
File Sharing System
User
Client
Index
Peer
Idle
Idle
download file
get peers
list of peers
LOOP n
T
set timer
get chunk from peer i
ALT
received chunk from peer
T
reset timer
T
timeout
Still enough peers left to rebuild the file
T
timeout
Not enough peers left to rebuild the file
abort
Idle
Rebuild file and save
Idle
Idle


MSC-004: Deleting a file
File Sharing System
User
Client
Index
Peer
Idle
Idle
delete file
get peers
list of peers
LOOP n
T
set timer
delete file
ALT
received delete file from peer i
T
reset timer
T
timeout
Still enough peers
to successfully
delete the file
T
timeout
Not enough peers
left to successfully
delete the file
abort
Idle
delete index for file
index deleted
Idle
Idle


MSC-005: Starting the Index component
File Sharing System
Init
Index
Idle
Idle
start
COND
Directory and Index File already exists
Load Index file
Directory already exists
Create Index file
Directory does not exists
Create Directory
Create Index file
Idle
Idle

Test Strategy

To ensure that the design worked and the implementation needs to be tested. The system is a fairly simple design so only a few test cases need to be considered. Each scenario should be tested to ensure that the main functionality is working. The alternative scenarios should also be tested to ensure that all errors are handled correctly.

Each of the following test cases should be tested with both IDA algorithms

Test Case 1: n = 4, k = 2
- upload a text file
- make sure the 4 peers each have a piece
- download the file and compare to the original

Test Case 2: n = 4, k = 2, one peer is not accessible at download time
- upload a text file
- make sure the 4 peers each have a piece
- close the shell on one of the peers
- download the file and compare to the original

Test Case 3: n = 4, k = 2, three peers are not accessible at download time
- upload a text file
- make sure the 4 peers each have a piece
- close the shell on three of the peers
- download the file, should generate an error

Test Case 4: n = 4, k = 2, only three peers are accessible
- upload a text file, should generate an error since there are only 3 peers

The following test cases are to test the index files

Test Case 5: Hard drive does not have read access
- remove read privileges from hard drive
- start the system, should generate an error

Test Case 6: Hard drive does not have write access
- remove write privileges from hard drive
- start the system, should generate an error

Test Case 7: Share directory does not exist
- ensure the share directory does not exist
- register some space for the share directory
- check to see if directory and share.dat was created

Test Case 8: share.dat does not exist
- ensure the share.dat in the share directory does not exist
- register some space for the share directory
- check to see if share.dat was created

Test Case 9: Test for both SIDA and IDA
- upload a file (using any values of n and k)
- check to see if the index file was written properly

Design Decisions

Design Decision: DD-001 Information Dispersal Algorithm (IDA)

A requirement of the system is that if a random number of peers are not accessible, the file should be completely recoverable. The only way to ensure this is to store the file on every peer and even then, if no peers are accessible, the file cannot be recovered. As discussed in Chapter 4, there are a few ways of storing a file on n peers such that if a number of peers are accessible (say k), the file can be recovered and the space requirement is dependent on the choice of k. For this system, both the SIDA and Rabin's IDA is implemented.

Design Decision: DD-002 Index File Format

After a file has been uploaded to the n peers, an index file must be maintained which indicates how the file can be retrieved. Since there are two algorithms being considered in DD-001, there must be two different formats for the index file. An index file is a file stored in the same directory as the shell application and has the filename index.dat.

A blank line in the index file delimits the files. The first line of the entry indicates the algorithm that was used to store the file (SIDA or IDA) and the chosen values of n

and k. If an encryption key and/or hash code was used, the key and/or hash code should be after the values of n and k. The remaining part of the format will be completely dependent on the algorithm.

SIDA

Each following line indicates what peer stored the file and the pieces that were stored at that peer. Here is an example of an SIDA index entry:

```
SIDA 4 2
key 35
hashcode 23452
peer1 1 2 3
peer2 2 3 4
peer3 3 4 1
peer4 4 1 2
```

IDA

Each following line indicates what peer stored the file and the key used for that peer. Here is an example of an IDA index entry: Indicies

```
IDA 4 2
key 35
hashcode 23452
peer1 [1 0]
peer2 [1 1]
peer3 [1 2]
peer4 [1 3]
```

Design Decision: DD-003 Space Registration

Before a user can upload a file, the user must first allocate space on the local machine. This is done to prevent the case where every user wants to upload files but no user has allocated space for file sharing. When a user registers space, a directory called "shared" is created from the running directory and a file is created called share.dat in the directory. This file contains information about the shared space. The file indicates how

much space has been allotted for the sharing and how much space has been used. Every time a file is uploaded or deleted this file gets updated.

The file contains two lines. The first is the size of the space allocated for sharing and the second is the total space used. Here is an example of share.dat:

```
1048576
536215
```

Security Concerns

Security Concern: SC-001 File confidentiality

It is important that an unauthorized user cannot view the file. For the current iteration, the only authorized user is the user who uploads the file. In future iterations, it is possible and should be considered that a file can be shared by a group of users.

It is easy to protect the file for confidentiality purposes. The file can be encrypted before it is sent using any secure encryption protocol. The most secure would be the One-Time Pad but any modern algorithm will do.

Security Concern: SC-002 Index File security

It is important that the index file is not corrupted. If it is, the file referenced by the index file will be lost. Since one of the main reasons of using the IDA algorithm is to ensure that a file cannot be lost if more then one piece is lost, it would be unacceptable to allow for the file to be lost if the index file is lost.

One solution to this problem is to use the IDA to store the index file on n peers. This way, the probability of the index file being unrecoverable is the same as the probability that the file being unrecoverable. This solution poses one problem. An index file would need to be stored locally to retrieve the index file stored using the IDA which brings us back to the original problem.

Another solution is to use Error Correcting Codes (ECC) on the index file to ensure that the file has not been corrupted. We can use an ECC that will not reduce the probability of recovering the file. The only problem remaining is losing the index file completely. The index file should be backed up to a safe medium to ensure protection of the files. Since the index file is relatively small in comparison to the files being stored, a fairly small backup device (such as a floppy or zip disk) can be used to backup the index file(s).

Security Concern: SC-003 Viruses

Since we are allowing files to be transmitted, we have to concern ourselves with viruses. If a peer receives and stores a file blindly that contains a virus, the virus can do damage to the peer's machine and may spread itself to other peers. There are two ways that this can be prevented.

The first is to compress the data as it is being received. Typically compressed data cannot be executed to perform a useful task. The second way is to encrypt the data as it is coming in. Encryption converts data (in this case a virus) into what appears to be random data. The only way the encrypted data can become the virus again is if it is decrypted. The only time that this happens is when the file is sent back to the user. The encryption algorithm can be as simple as XORing the data with a random key. Performing the XOR operation using the same key will decrypt the file.

Security Concern: SC-004 File Integrity

Both SIDA and IDA are capable of rebuilding a file if there are at least k pieces. The algorithm does not guarantee if a peer was tampered with, the file can still be recovered (or at the very least an error should be reported). There are two ways for finding an error with both algorithms.

The first is to use the property of the SIDA and IDA. Only k pieces are required to rebuild the file. This means, if k+1 pieces can be downloaded (if $k \neq n$), the file can be rebuild using any k of these chunks. There are k+1 different ways of reconstructing the file. If any two reconstruction of the files are different, then we know that at least one piece can be tampered with. If only one peer was tampered with, it would be easy to find out which one.

In the SIDA algorithm, it would not be hard for an attacker to determine which piece is which and tamper with all of the pieces on all of the peers such that even using k+1 pieces the file would be considered tampered free. If we were to encrypt all data sent to a peer and decrypt it on the way back, an attacker could not coordinate its attack on all of the peers. The IDA algorithm does not have the same problem as the SIDA since it uses a key, which is unknown to the attacker to disperse the file. Without the key, the attacker could not coordinate its attack on all the peers.

The second way of finding an error is to store the one-way hash of the file (or a CRC or MAC code) and when the file is recovered it can be compared to the hash to check for tampering. This way is easier then the one above but it does not help find which peer was tampered with. If both of these ways are combined, we can check for errors and attempt to correct the errors.

6 Conclusions

This report discusses in detail how one would implement a file sharing application using an information dispersal algorithm to reconstruct the algorithm even when some of the peers are not accessible. There are many improvements one could apply to this design such as peer authentication, or support for peer groups with the IDA (allowing all peers in the group to have access to the index file).

I hope that this report is both interesting and useful for all who reads it as I found this topic interesting and informative.

7 References

[1] A. Bestavros, *SETH : A VLSI Chip for the Real-Time Information Dispersal and Retrieval for Security and Fault-Tolerance*, Department of Computer Science, DEAS, Harvard University, (August 1990)

[2] M. Rabin, *Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance*, Technical Report, TR-02-87, Department of Computer Science, DEAS, Harvard University, (April 1987)

[3] F. Bordeleau, *A Systematic and Traceable Progression from Scenario Models to Communicating Hierarchical State Machines*, PhD. Dissertation, Carleton University, Faculty of Engineering, Ottawa, Canada, (1999)

Web Sources

http://www.zdnet.com/zdnn/stories/news/0,4586,2835075,00.html

http://www.zdnet.com/zdnn/stories/news/0,4586,2803865,00.html

http://www.caip.rutgers.edu/~vincentm/p2p.html

http://members.fortunecity.com/pcmuseum/windows.htm