HYBRID REPLICATING PEER-TO-PEER SYSTEM

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING
OF

BHARATHIAR UNIVERSITY
COIMBATORE

SUBMITTED BY

KARTHIK. K

MURUGESH. K

RIMNA. R

ZULFIKAR Z. IMANI

UNDER THE GUIDANCE OF

Mrs. N. CHITRA DEVI M.E., LECTURER
DEPARTMENT OF COMPUTER SCIENCE,
KUMARAGURU COLLEGE OF TECHNOLOGY,

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

MARCH-2003

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

PROJECT REPORT 2002-2003

This is to certify that the project report entitled

HYBRID REPLICATING PEER-TO-PEER SYSTEM

is the bonafide work of

Karthik K.	**9927K0128**
Murugesh K.	**9927K0142**
Rimna R.	**9927K0156**
Zulfikar Z. Imani	**9927K0176**

in partial fulfillment of the requirement for the award of **Bachelor's Degree in Computer Science and Engineering** of Bharathiar University, Coimbatore – 641042

Project Guide

Head of Department

Place: COIMBATORE

Date: 18th MARCH 2003

Submitted for the university exam held on 18th **March 2003**

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We wish to express our sincere gratitude to Dr. K. K. Padmanabhan, Ph.D., Principal, Kumaraguru College of Technology, for providing us with the impetus and infrastructure for doing the project.

We would like to thank Dr. S. Thangasamy, Ph.D., Head of the Department, Computer Science and Engineering, for the support and interest he showed in our project.

We would also like to thank our project coordinator, Ms. S. Rajini, B.E, Senior Lecturer, Computer Science and Engineering Department. She played a major role in contributing towards the project success.

Our deepest gratitude goes to our project guide Ms. N. Chitra Devi, M.E, Lecturer, Department of Computer Science and Engineering, for the encouragement and support she gave us during the project. Her invaluable suggestions coupled with her technical acumen provided us with the right background to work on.

This acknowledgement would be incomplete if we do not mention about the support rendered to us by our Class Advisor Mr. M. N. Guptha, Lecturer, Department of Computer Science and Engineering, whose constant encouragement and advice guided us in the right direction.

Finally we take this opportunity to thank all our friends for their constant encouragement and all the teaching and non-teaching staff without whom this project would not have been a success.

PROJECT PROFILE

TITLE : **HYBRID REPLICATING PEER-TO-PEER SYSTEM**

DEVELOPED AT : KUMARAGURU COLLEGE OF TECHNOLOGY

PURPOSE : To develop a robust peer-to-peer system incorporating replication for developing a reliable backbone which is augmented with services such as file transfer,remote computing, messaging , file searching.

SOFTWARE USED : Java™ Development Kit, version 1.2, Java2 Platform

HARDWARE USED : Pentium III 700MHz with 64MB RAM and 20GB HDD

INTERNAL GUIDE : Mrs. N. Chitra Devi M.E., Lecturer,
Department Of Computer Science & Engineering,
Kumaraguru College Of Technology, Coimbatore.

SUBMITTED TO : The Department Of Computer Science & Engineering,
Kumaraguru College Of Technology, Coimbatore.

SUBMITTED BY :

Karthik. K	- 99CSE16
Murugesh. K	- 99CSE30
Rimna. R	- 99CSE44
Zulfikar Z. Imani	- 99CSE64

SYNOPSIS

Our project entitled "**Hybrid Replicating Peer-to-Peer System**" was mainly taken up keeping in mind the bottlenecks and limitations of centralized Client/Server systems wherein certain centralized nodes called servers consist of huge amount of information and processing power from which the connected clients can request. The major drawbacks of centralized systems are over-dependence on central nodes or servers which may bring an entire network to a halt if the server crashes, complex load balancing and fault tolerance algorithms are required to keep the servers working effectively and fully functional and centralized systems are vulnerable to attacks. Our project mainly focuses on establishing a decentralized network wherein parts of the system or even the whole system are no longer operated centrally. Decentralization is in particular interesting in order to avoid single-point-of failures or performance bottlenecks in the system. Our project mainly aims at building such a decentralized system or a peer-to-peer system (P2P system) in which each node or peer acts as both a client and a server and pays its participation by providing access to its computing resources. When a P2P system becomes fully decentralized then there no longer exists a node that can centrally coordinate all activities or a database to store global information about the entire system centrally. Therefore nodes have to self-organize themselves, based on whatever local information is available and interacting with locally reachable nodes (neighbors). So the system will be designed keeping in mind the above complexities of a de-centralized system. Suitable communication protocols along with appropriate features will be incorporated in order to ensure proper sharing of resources and computational power. We plan to enhance the system by incorporating the idea of replication to make the system more reliable, which will allow the nodes to maintain a reliable communication even in an unreliable environment. Once the P2P-backbone is built, major modules like file sharing and remote computing can also be incorporated in the application layer of the P2P system.

Below stated is an adhoc description of the major modules of the project:

Building the replicating peer-to-peer system:

This will be the major system, which will form the backbone on which the other modules of the project will be built. This will involve developing the decentralized P2P system using appropriate protocols and security features to ensure an effective and robust communication system.

File Sharing:

This module will aim at incorporating an effective file access and transfer mechanism including advanced file search features using which each node in P2P framework will have access to the shared files of the other nodes. Peers can download files from the other peers thus enabling efficient resource and information sharing.

Remote computing:

This module will allow nodes to send certain specific jobs to any one of its peers for processing. The peer, which has the particular resource to process the job, gets the job processes it and sends back the results to the requestor. The allocation of the job to the appropriate peer is done using suitable protocols.

CONTENTS

1. INTRODUCTION.....2
   1.1 OBJECTIVE
   1.2 SCOPE
2. LITERATURE STUDY.....4
   2.1 NAPSTER
   2.2 GNUTELLA
   2.3 FREENET
3. SYSTEM REQUIREMENTS.....7
   3.1 PRODUCT DEFINITION
   3.2 PROJECT PLAN
4. SOFTWARE REQUIREMENTS SPECIFICATION.....13
   4.1 PURPOSE
   4.2 SCOPE
   4.3 PRODUCT OVERVIEW AND SUMMARY
   4.4 EXTERNAL INTERFACES AND DATA FLOW
   4.5 FUNCTIONAL SPECIFIACTION
   4.6 EXCEPTION HANDLING
   4.7 ACCEPTANCE CRITERIA
5. SOLUTION STARTEGY.....19
6. DESIGN SPECIFICATIONS.....22
   6.1 SCENARIOS
   6.2 DATA FLOW DIAGRAMS
7. IMPLEMENTATION DETAILS.....29
   7.1 ALGORITHMS
8. TESTING.....34
9. FUTURE ENHANCEMENTS.....37
10. CONCLUSION.....40
11. BIBLIOGRAPHY.....42
12. APPENDICES.....44

INTRODUCTION

1.1 OBJECTIVE

In current file sharing environments we mostly come across centralized systems where a main server is at the hub of the activity with all the resources concentrated on it. To overcome this we plan to develop a novel-networking environment similar to a P2P system, which harnesses the resources of each individual node (peer) and does not depend excessively on just a single node for file and resource sharing. Existing P2P systems have the inherent disadvantages like lack of reliability and stability. Organizing nodes in a P2P system requires complex protocols and is not totally error free.

Our project mainly concentrates on incorporating reliability through the process of replication wherein the files located in each node are replicated or copies of it are written across several peers logged on to the network. Organization of peers is done using effective communication protocols. The user is provided with essential services like file sharing and searching facilities, remote computing and instant messaging through a user-friendly interface.

1.2 SCOPE

Apart from being deployed over an intranet or the World Wide Web this product can also be used in corporate houses, who want to have a P2P communication system built on top of their existing inter network without having to employ specialized personnel to handle and maintain centralized systems. Users can share a pool of resources instead of having to upload the same on centralized servers, which are more vulnerable to network contingencies.

LITERATURE STUDY

2.1 NAPSTER

Napster is a popular example of a P2P system widely used for sharing songs over the Internet. Napster mixes centralization and decentralization as a part of its architecture. It builds and maintains a master list of the resources, adding and removing resources as individual users connect and disconnect from the network. This master list of resources can be accessed through a server. Once the presence of a particular resource is identified, the server, redirects the file transfer to take place between the requesting and supplying peers.In this particular case a widely evident loophole is that if a particular directory server goes down then the entire system crashes. So reliability and stability of the system is compromised.

2.2 GNUTELLA

Gnutella is example of a completely decentralized system. Here both the search for the resources and the interaction between the peers are essentially peer-to-peer. In this system the user logs into the network by specifying the name of a well-known host who has already logged into the system or this information is made available by using a set of host caches. When a peer requests for a particular resource on the system this request is routed to the peers it that route it subsequently to other peers they have encountered and so on. The request is broadcasted by a particular peer to all the peers it has encountered so far. The inherent disadvantage of Gnutella is that since new peers make use of host caches to enter the network there is a high probability of the network being restricted to a few tightly clustered nodes instead of encompassing the entire network since the host caches only carry information about the peers that they have seen recently.

2.3 FREENET

Freenet is a decentralized system for distributing files that demonstrates a particularly strong form of P2P communication. Here requests are forwarded as a chain of messages by peers to the peers that they completely trust. Messages time out after passing through a certain number of nodes, so that huge chains don't form. The chain ends when the

message times out or when a node replies with the data. The reply again passes through all the nodes that forwarded the file request. All the nodes in the chain inorder to improve the reliability of the system may do caching. The drawback of such a system is that it is difficult to determine how many nodes are actually present in the network, hence difficult to organize.

SYSTEM REQUIREMENTS

3.1 PRODUCT DEFINTION

3.1.1 PROBLEM STATEMENT & SYSTEM JUSTIFICATION

Studying the characteristics of a centralized Client/Server system wherein centralized nodes called servers consist of huge amount of information and processing power from which the connected clients can request, we found certain major and critical loopholes and drawbacks in the system. Over-dependence on central nodes or servers may bring an entire network to a halt if the server crashes, complex load balancing and fault tolerance algorithms to keep the servers working effectively and vulnerability to attacks. So we propose to develop a fully decentralized network wherein parts of the system or even the whole system are no longer operated centrally. Our project mainly aims at building a peer-to-peer system (P2P system) in which each node or peer acts as both a client and a server and pays its participation by providing access to its resources hence overcoming the major drawbacks of a centralized system.

3.1.2 FUNCTIONAL SPECIFICATION

The aim of the project is to develop a robust and reliable backbone, which provides an effective resource-sharing environment. It should facilitate the user to share files among various other users who have logged in to the network. It should provide the functionality, which enables a user to search for required resources and make use of them efficiently. File resumption capability should be a inherent part of the system which enables the user to overcome unforeseen situations. This system should also incorporate remote computing facility using which the computing resources of various peers can be utilized in an effective manner.

3.1.3 DEVELOPING AND PROCESSING ENVIRONMENT

The Java™ Runtime Environment 1.2 (JRE) is the Java platform on which we are going to develop our application. It consists of the Java virtual machine, the Java platform core

classes, and supporting files. Java™ Development Kit, version 1.2 is going to be used for development, compilation and creating executables. Since Java compiles the source files into byte code and generates class files, which can be run on any platform, peers can operate this product on any platform, which has the appropriate JVM. The JVM interprets and runs the class files or a JIT compiler, which translates the byte code to native code, can be used.

Typical operating platform requires OS which supports GUI, 486MHz or higher end processors with minimum 32MB RAM, Hard Disk Capacity of 2GB or more to support replication.

3.1.4 SYSTEM CONSTRAINTS

- Since P2P systems are generally based on point-to-point communication, our system when deployed on broadcast networks results in overheads like wastage of bandwidth, additional traffic etc.
- Amount of resource that can be replicated on peers cannot exceed a threshold limit. Hence reliability is compensated for overhead and system complexity.
- In order to strike a balance between efficiency and queuing space the number of simultaneous requests is restricted to a threshold value.
- For easy protocol design and management we plan to have peer groups with a maximum of 256 peers in each group (Class C network). System is not extensible for Class A & B networks.

3.1.5 USER CHARACTERISTICS

The user of this product should have a basic knowledge of using a GUI. Since the project deals with File Sharing, Remote Computing in a network, basic idea of networking will enhance the usability of the system. Novice as well as experts can benefit from the simple design of the system.

3.1.6 PRODUCT FEATURES

The product provides a suitable communication environment, which the users can employ to share resources, search for files, start chat sessions and communicate effectively in a reliable network. The notable feature of this system is that even in an unreliable environment where peers do not have any specific trend of network usage this system is able to provide certain amount of reliability through replication of files across the various logged in peers. The front end is designed keeping in mind the different levels of expertise of user. It provides an interactive interface, which easily enables a user to select his option and carry out any transaction with ease. The product is also portable across any platform by making minor changes.

3.1.7 GLOSSARY OF TERMS

HRPP - Hybrid Replicating Peer-to-Peer System

P2P - Peer to Peer

GUI - Graphical User Interface

RAM - Random Access Memory

MHz - Mega Hertz

GB - Giga Byte

OS - Operating System

JDK - Java Development Kit

JIT - Just In Time

JRE - Java Runtime Environment

JVM - Java Virtual Machine

MB - Mega Byte

HDD - Hard Disk Drive

3.2 PROJECT PLAN

- The **life cycle model** for our project will be a simple **SDLC** consisting of Analysis, Design, Coding and testing phases.

- **Team structure** will be **democratic** in which every member will contribute equally to the project development.

- Various documents giving a detailed view about the various milestones reached during the course of the project are to be prepared along with the detailed documentation of the analysis, architectural and detailed design with the implementation details.

3.2.1 LIFE CYCLE MODEL

PLANNING

- **Milestones** : Nov 20th to 22nd 2002
- **Activities** : Risk analysis was done and a tentative schedule was decided upon, the software tools and languages to be used were determined and understood.

ANALYSIS

- **Milestones** : Nov 25th to 30th 2002
- **Activities** : Analysis of the product definition . The functions and features the product has to perform, the processing environment were decided upon. The requirements of the project were enumerated and a detailed functional specification of major subsystems were embodied into the SRS.

DESIGN

- **Milestones** : Dec 1st to Dec 25th 2002
- **Activities** : The design of the protocol stack to be used in several layers of P2P communication system. Identifying the major subsystems of the project. Determining the various scenarios that could result and representing the same using high level DFD's and state diagrams.

CODING

- **Milestones** : Dec 27th 2002 to Feb 20th 2003
- **Activities** : Deciding upon the functions to efficiently implement the protocol. Designing the interface between the subsystems and integrating them implementing the various data structures necessary. Front end design and creation.

TESTING

- **Milestones** : Feb 23rd to Feb 25th 2003
- **Activities** : The modules will be individually tested for correctness and completeness and then integrated and tested for interface compatibility and the prototype version will be deployed over a Class C network and finally the acceptance criteria will be checked for.

SOFTWARE REQUIREMENTS SPECIFICATION

4.1 PURPOSE

The primary purpose of the Software Requirements Specification (SRS) is to document the previously agreed to functionality, external interfaces, attributes, and the performance of the HRPP system. This specification is the primary document upon which all of the subsequent design, source code, and test plan will be based

4.2 SCOPE

The scope of this document, Software Requirement Specification (SRS) is to describe the requirements definition effort. The SRS documentation for the HRPP System describes the functions, external interfaces, attributes, and performance issues specified in the product definition.

4.3 PRODUCT OVERVIEW AND SUMMARY

Our product provides a reliable, robust and efficient means of establishing a P2P communication system. Our product uses complex replication strategies to overcome the inherent disadvantage of P2P systems by improving the reliability. The application layer of the system consists of efficient file searching and sharing strategies along with facilities for remote computation. This product can be used in corporate houses, which want to have a P2P communication system built on top of their existing inter network without having to employ specialized personnel to handle and maintain centralized systems. Users can share a pool of resources instead of having to upload the same on centralized servers, which are more vulnerable to network contingencies.

4.4 EXTERNAL INTERFACES AND DATAFLOW

USER DISPLAY & REPORT FORMATS

The external user interface will enable the user to become a part of the peer group and get details of the other peers logged in the same group, select the files and copy them into a

shared folder, register the user's profile, search for files based description, type and name, options for file transfer and remote computing will be provided as well as status reports on downloads and other information.

4.5 FUNCTIONAL SPECIFICATION

4.5.1 REPLICATING BACKBONE

4.5.1.1 Description

This module gives the system the required reliability and the basic communication backbone. This existence of multiple copies of the same file ensures reliability and also increases the efficiency. The protocol suite is designed in a manner so as to satisfy the basic communication requirements, login and logout sessions.

4.5.1.2 Criticality

Since this sub-system forms the foundation for the underlying sub-systems and application frame works , its stability and efficiency are most important.

4.5.1.3 Technical Specification

LOGIN

Communication is established using sockets between connecting peers. Each peer should be capable of sharing some of its storage resource for replication of files. The backbone is designed in such a manner that whenever a peer logs in all the files which he is willing to share are replicated across different peers using a specified protocol. In order to give each peer an equal chance of replicating its files, a constraint, which limits the amount of information that can be replicated, is enforced.

LOGOUT

Whenever a peer logouts of the network and its file is being requested by another peer the logging out peer should reroute the request to another peer who has the replicated copy of that file. This enables the requesting peer to resume the file transfer transparently. To free up the storage space and to use it effectively, every time a peer logs out its replicated files are deleted and peers who have replicated their files in the logged out peer should repeat their replication cycle.

APPLICATION SERVICES

The replicating backbone is augmented by the following application services:

4.5.2 FILE SHARING AND SEARCHING

4.5.2.1 Description

This sub-system forms the application framework over the basic replicating backbone. They provide the basic application service like file searching, sharing and allow peers to upload and download files from other peers. As an option to use the storage resources of the peers effectively we propose to have certain mechanisms.

4.5.2.2. Criticality

It provides the basic service required by a user hence has to be robust, reliable and easy to use.

4.5.2.3 Technical Specification

Request for a particular file (name or description) is broadcasted to all connected peers. The peers, which have the file, respond with a positive acknowledgement. The node from which the file has to be transferred is left to the user's discretion. One of the mechanisms to

use storage space effectively could be the maintaining of an access counter for each file, which is incremented whenever a request for a file comes. Based on this value of the counter multiple copies maybe replicated to balance the load across peers or maybe deleted to free up the space.

4.5.3 REMOTE COMPUTING

4.5.3.1 Description

This module enables the user to use computing resources present in several peers across the network. The user searches for a resource, establishes connection with the peer, which has the required resource, sends the input, which is remotely processed and receives the output.

4.5.3.2 Criticality

Since it is just an added application service the criticality of this module is not very high.

4.5.3.3 Technical Specification

Request for a particular resource (name or description) is broadcasted to all connected peers. The peers, which have the resource, respond with a positive acknowledgement. The user decides to which peer he wants to send the input for computation. The remote peer, who has the resource accepts the input, generates the required result and sends it back to the requestor in the form of a file. The remote processing is transparent to the peers and the interface aids the peers to accept or reject the requests of several peers.

4.5.4 INSTANT MESSAGING SERVICE

4.5.4.1 Description

It's a simple mechanism using which peers can have a one-to-one or one-to-many communication.

4.5.3.2 Criticality

Since it is just an added application service the criticality of this module is not very high.

4.5.3.3 Technical Specification

This module gives the user a list of all logged in peers, which is obtained, at regular intervals from the server process, from which the user can select a single peer to message or can send his message to all the logged in peers.

4.6 EXCEPTION HANDLING

- If a particular file is not replicated and that file is being transferred between two peers and the peer which has the file logs out. Then a file resume option can be provided.
- If the number of simultaneous request for a particular file to a peer exceeds a threshold.Then the remaining request are re-routed to the peers which have the replicated file.

4.7 ACCEPTANCE CRITERIA

- It should provide a basic reliable service under variants of network traffic.
- Integrity of the system should be maintained in the network by means of consistent data sources.
- Transparency in data transfer should be provided.
- Equal opportunity to access shared information should be provided to all peers.

SOLUTION STRATEGY

The problem was approached in a systematic manner pertaining to the software development life cycle. We studied the domain of P2P systems and analyzed their characteristics. Out of the many loopholes and drawbacks we found, we made a detailed study on the reliability and how peers organize themselves. We decided to develop the core of the system keeping in mind the above two parameters. The requirements of such a system were thoroughly analyzed and discussed. Based on the system requirements the SRS was prepared followed by the detailed design of the architecture and the implementation of the prototype version. After each stage the guide approved the corresponding document before moving on to the next phase.

5.1 PHASE I

This phase will involve developing the backbone of the system on a prototype network consisting of limited number of peers with limited resources and then testing the same for inconsistencies and irregularities. We will concentrate mainly on developing the basic and essential protocols for peer login, replication and basic communication. Peer login will involve broadcasting a request over the network. All peers who are logged in acknowledge the peer's request and the peer stores the necessary details. Replication will involve creating multiple copies of the same file across different peers. Communication involves basic messaging and file transfer facilities.

5.2 PHASE II

In this phase we will augment the system by adding more peers and more resources on each peer and verify the efficiency of the protocol design. The logout subsystem will be developed which makes sure that the storage space of all peer is utilized efficiently by freeing it up and proper signals are given to the peers so that the consistency is maintained.

5.3 PHASE III

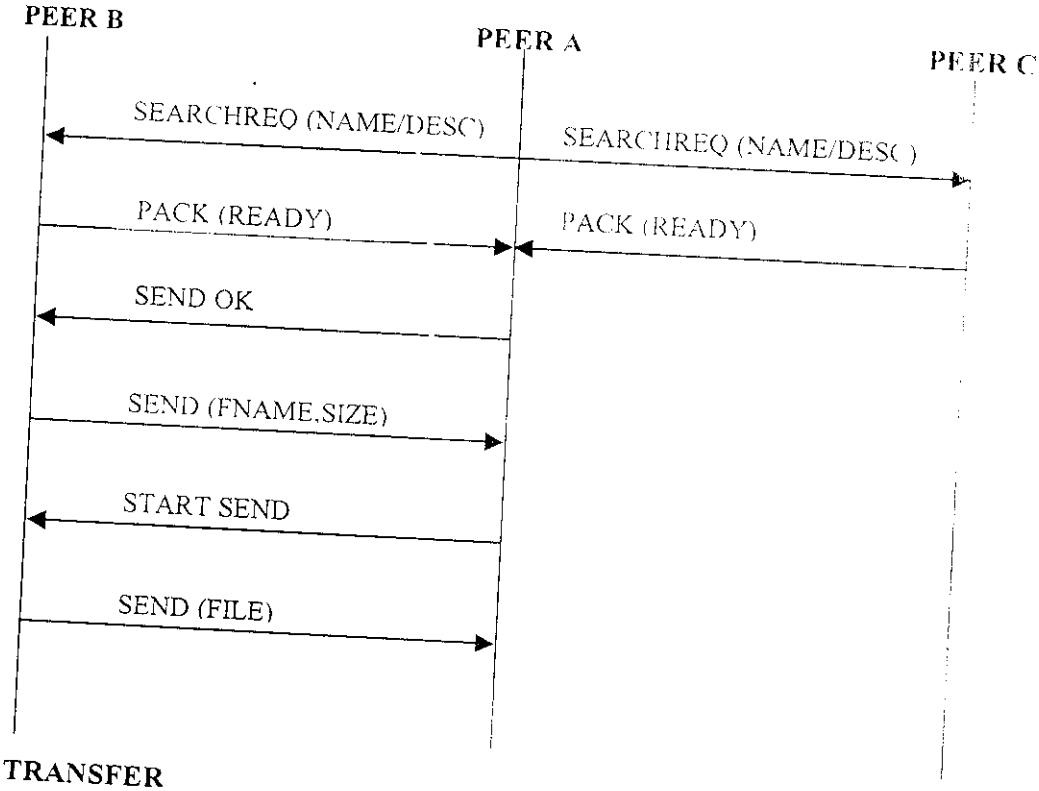
The main objective in this phase will be to add application services like file searching, sharing and remote computing on the developed backbone. The efficiency of the replication subsystem will be checked using certain predefined scenarios. In this phase the intranet communication will be fully validated by testing the application services. These application services are incorporated mainly to help peers overcome their inherent disadvantages like lack of computation resources. The file searching mechanism should enable a peer to search a file based on either the description or the name. It is left up to the user to select from which peer he wants to transfer the file. In a similar manner a remote computing facility will be developed using which a peer searches for a computing resource and passes on the necessary parameters which are remotely executed and the results returned back to the peer.

DESIGN SPECIFICATIONS

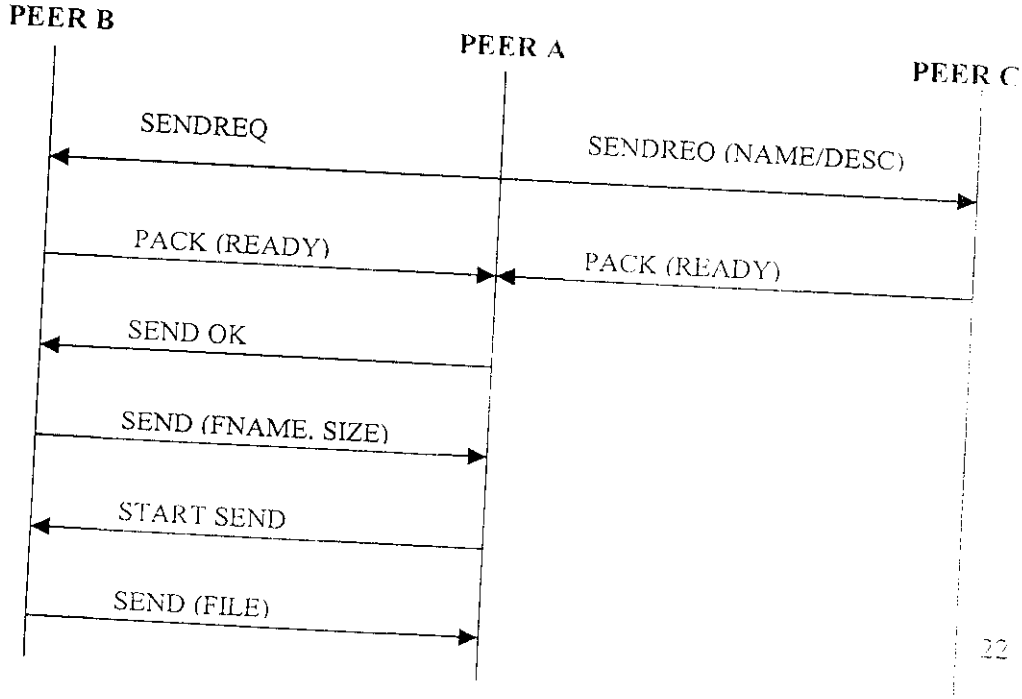
The following design documents are produced:

6.1 SCENARIOS

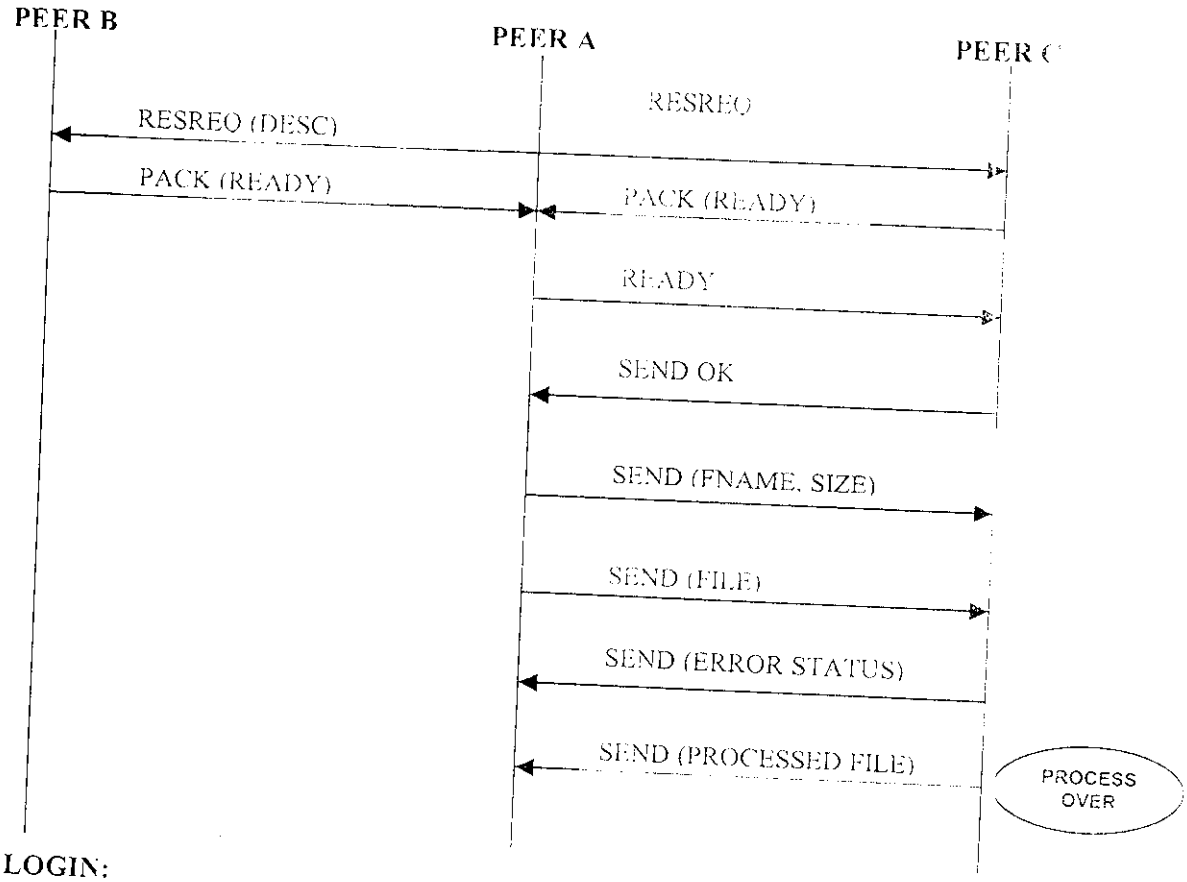
6.1.1 FILE SEARCH



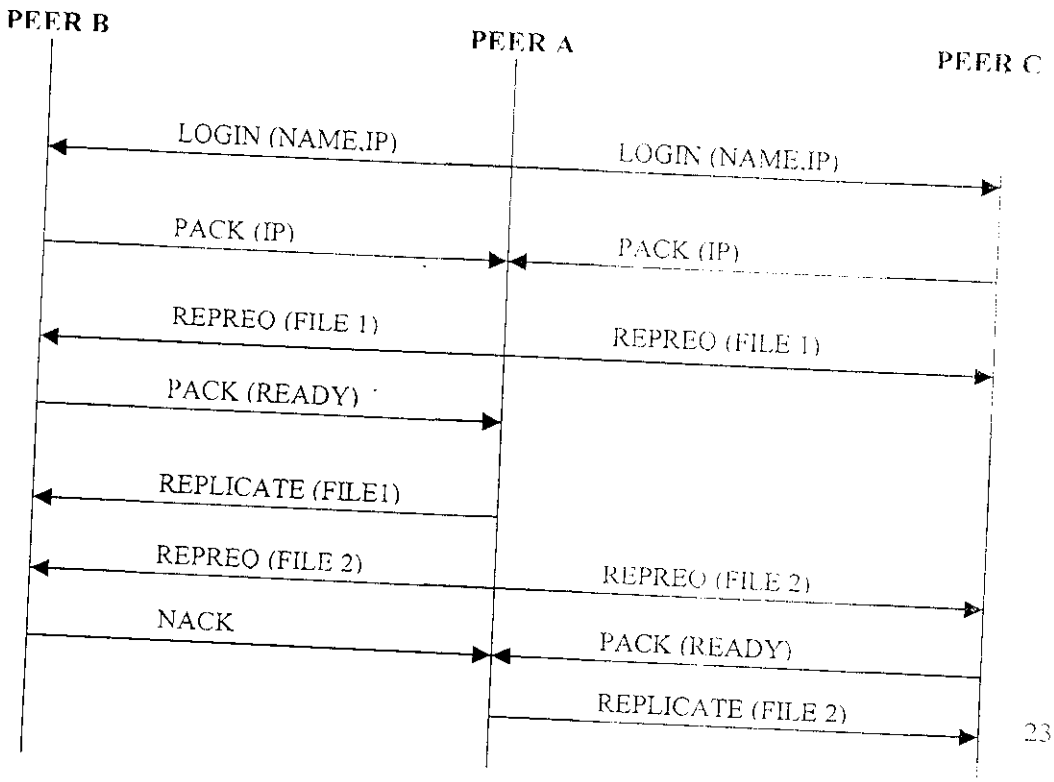
6.1.2 FILE TRANSFER



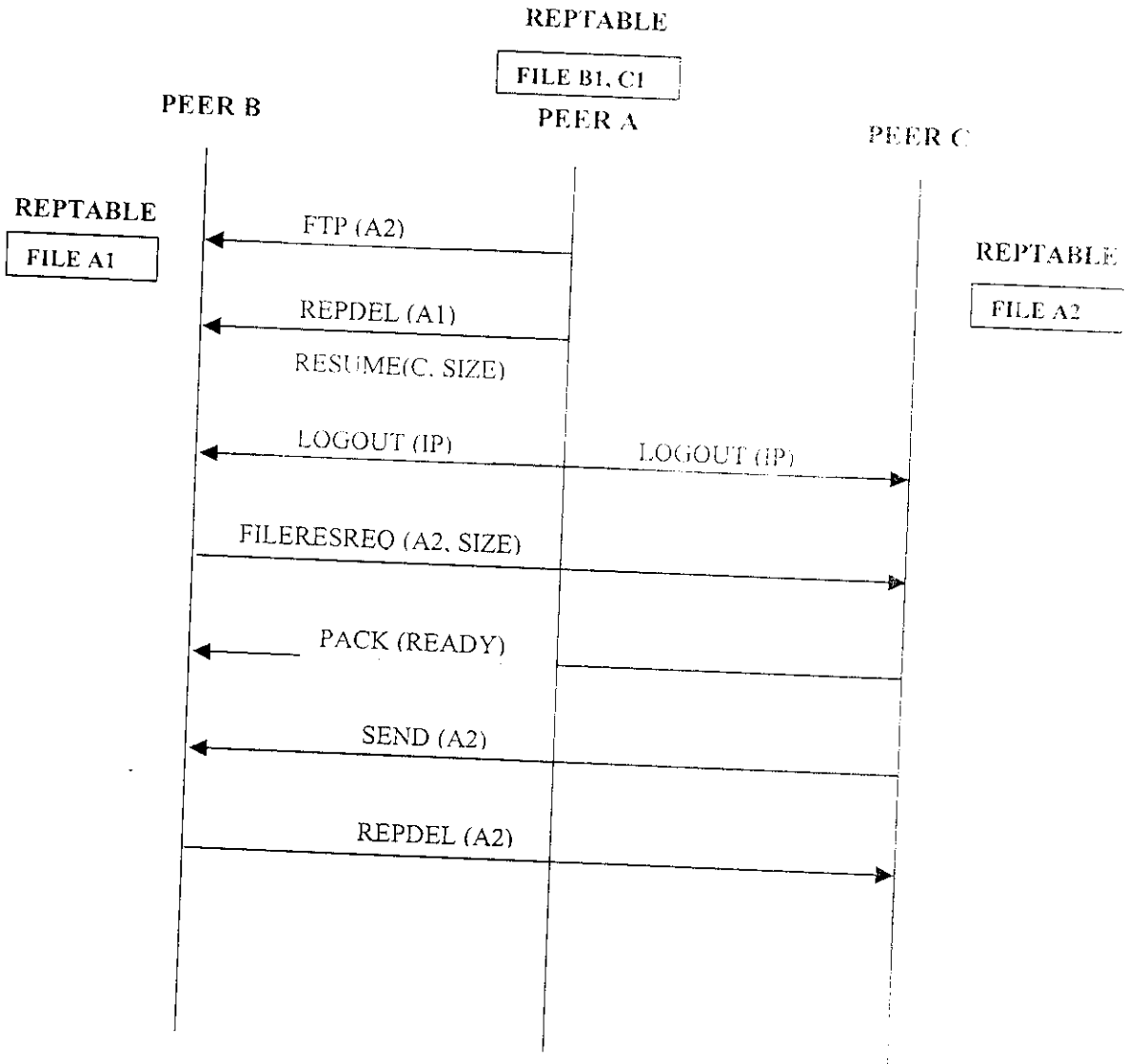
6.1.3 REMOTE COMPUTING



6.1.4 LOGIN:

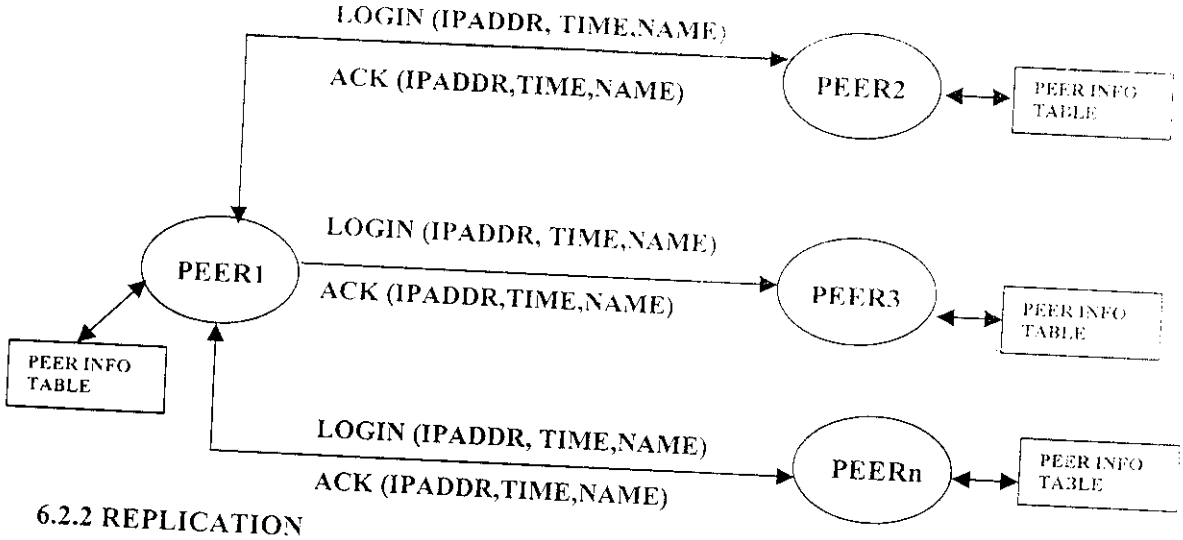


6.1.5 LOGOUT

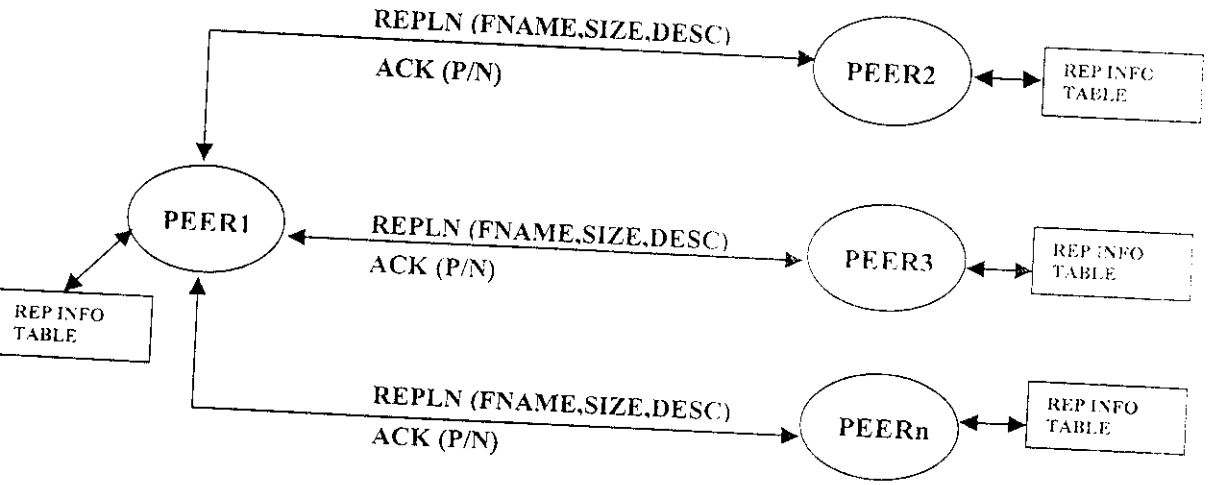

REPTABLE
FILE B1, C1
PEER B
PEER A
PEER C
REPTABLE
FILE A1
REPTABLE
FILE A2
FTP (A2)
REPDEL (A1)
RESUME(C, SIZE)
LOGOUT (IP)
LOGOUT (IP)
FILERESREQ (A2, SIZE)
PACK (READY)
SEND (A2)
REPDEL (A2)

6.2 DATA FLOW DIAGRAMS

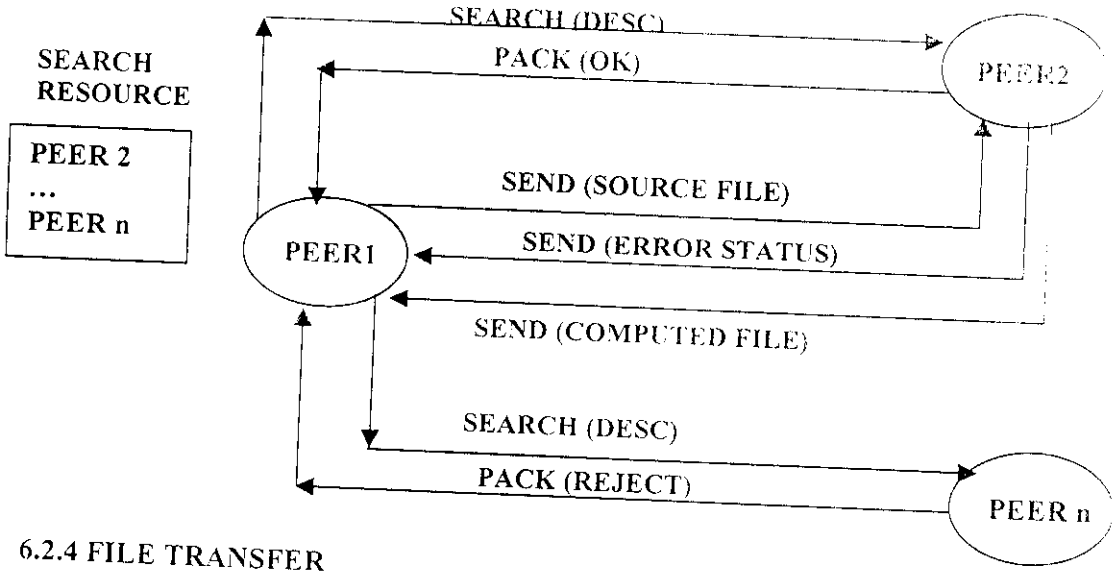
6.2.1 LOGIN INTERACTION



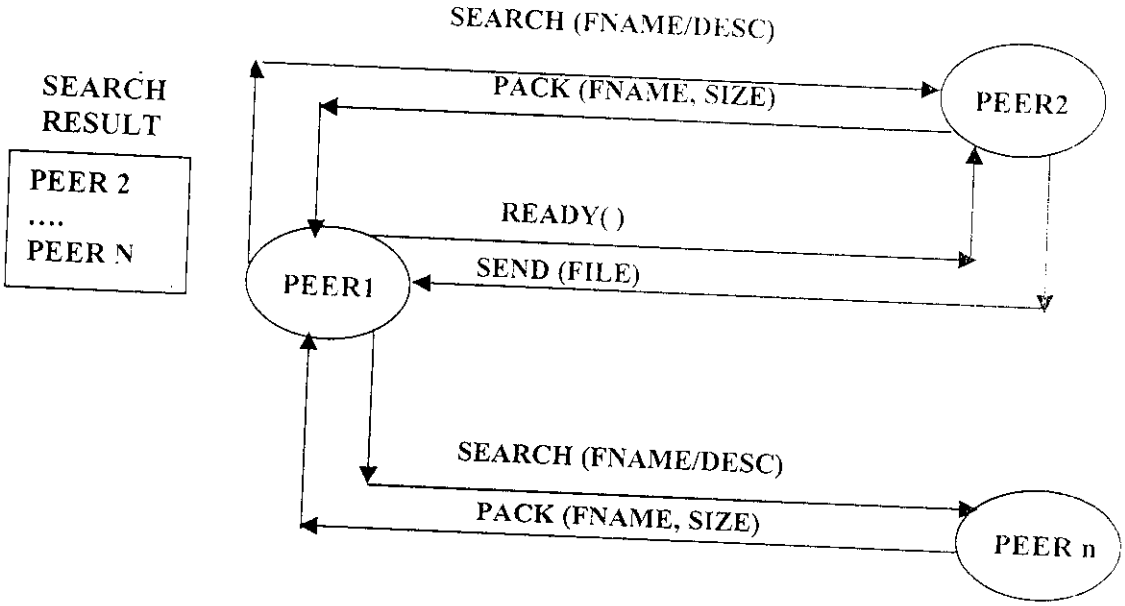
6.2.2 REPLICATION



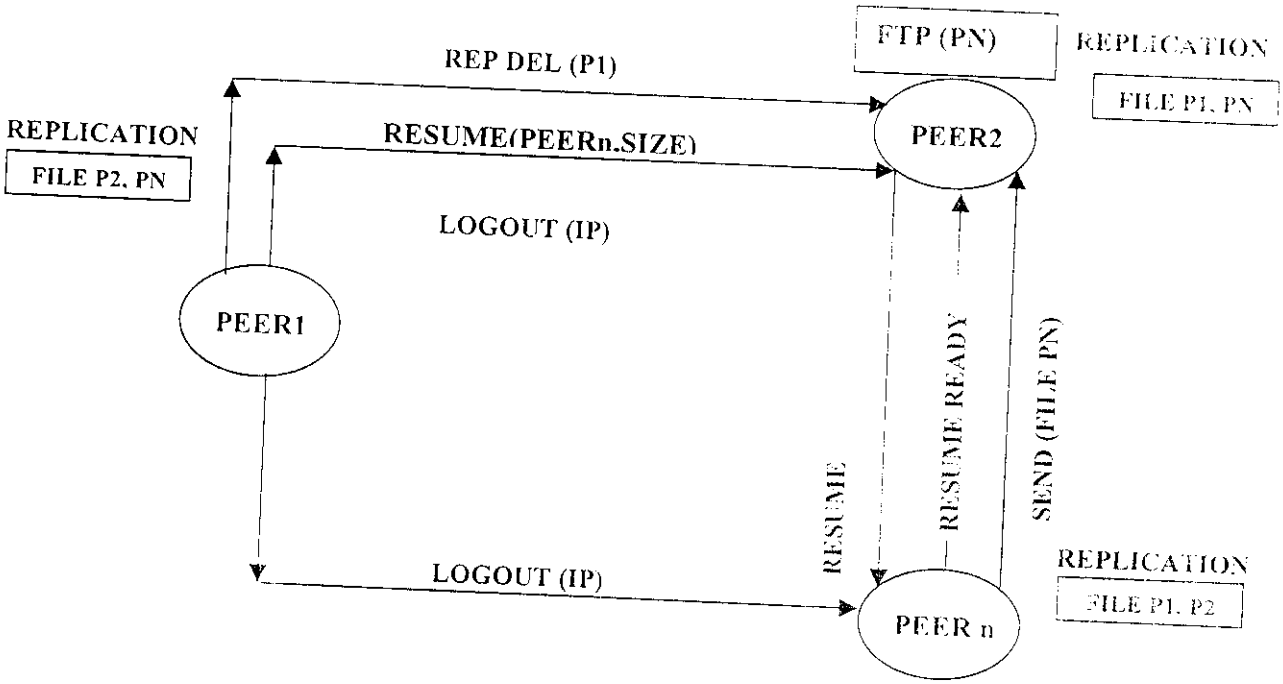
6.2.3 REMOTE COMPUTING



6.2.4 FILE TRANSFER



6.2.5 LOGOUT


FTP (PN)
REPLICATION
FILE P1, PN
REP DEL (P1)
REPLICATION
FILE P2, PN
RESUME(PEERn.SIZE)
PEER2
LOGOUT (IP)
PEER1
RESUME
RESUME READY
SEND (FILE PN)
LOGOUT (IP)
REPLICATION
FILE P1, P2
PEER n

IMPLEMENTATION DETAILS

7.1 ALGORITHMS

7.1.1 LOGIN

1. Build a datagram packet and broadcast it to all the peers in the network.
2. Construct a socket in a pre-defined port where the logged in peer listen for the acknowledgement form the other peers.
3. Receive the ack packet from al the peers in the network.
4. Store the IP address in the array.
5. Increment the array and repeat the process for a specific time, in order to wait for all the peers to respond.
6. Set up the files that are to be replicated.
7. Repeat for I = 1 to 'n' (no of files to be replicated)
8. Get the size of file to be replicated.
9. Broad cast the size of the file to be replicated to all the peers in the network.
10. Receive the ack packet from all the peers that have sufficient replication space.
11. Set up an ACK ARRAY to store the IP address of all the peers.
12. Create a REP ARRAY to store the file name and the IP address of the peer where it has been replicated.
13. Replicate the first file in any one of the acknowledging peers.

During subsequent replication processes check the REP ARRAY and the ACK ARRAY and using the details choose the peer, which has been, used for replication the least.

14. Send a READY packet to the selected peer for replication.
15. Initiate file transfer between the two peer by sending the file size and the name.
16. Continue the loop for n files.

7.1.2 FTP

1. Start a server socket that is listening at a predefined port at a remote peer where the file has to be transmitted.
2. Create a client socket at the user side and establish a connection between them.
3. Browse through the directories and select the file to be transmitted.
4. Send a packet from the user side with the name and the size of the file.
5. The server process acknowledges the packet by sending a SEND packet.
6. On receiving the SEND packet the user process converts the file into a byte stream and sends it to the server process.
7. The server process reads byte by byte and writes it into a file.

7.1.3 FILE SEARCH

Each peer configures the files to be shared with its description and name. These are stored in FILEDESC ARRAY and FILENAME ARRAY.

1. Get the choice for searching the file from the user, either the name or the description.
2. If the description is selected add DESC identifier or the NAME identifier if the name of the file is selected to the packet along with the search string.
3. Start a server socket that is listening at a predefined port at a remote peer where the file has to be transmitted.
4. Create a client socket at the user side and establish a connection between them.
5. The server process checks the entries in the FILEDESC ARRAY or FILENAME ARRAY according to the identifier present in the packet for the search string.
6. Ack Packet is sent to the user process with the size of the file from the server side.

7. The user process stores the ack packet received from different peers in an array.
8. Get the user choice on selecting the peer from which the file has to be transferred
9. Send a READY packet to the remote peer.
10. Initiate the file transfer using the FTP Entity.

7.1.4 REMOTE COMPUTING

Each peer configures the resources to be shared with its description .This is stored in the RESDESC ARRAY .

1. Get the user description of the resource to be shared.
2. Start a server socket that is listening at a predefined port at a remote peer where the file has to be transmitted.
3. Create a client socket at the user side and use it to broadcast requests over the N/W.
4. The server process checks the entries in the RESDESC ARRAY for matching resources at the remote peer side.
5. Ack Packet is sent to the user process from the server side.
6. The user process stores the ack packet received from different peers in an array.
7. Get the user choice on selecting the peer to which the file has to be sent for remote processing.
8. Send a READY packet to the remote peer.
9. Initiate the file transfer using the FTP Entity to transfer the source file to the remote peer where the resource is available.
10. The server process running in the remote peer receives the file and processes it and produces the output.
11. An output or an error report is sent to the requesting peer.

7.1.5 LOGOUT

1. Check the REP ARRAY to find where the files have been replicated.
2. An FTP ARRAY is created to record all the file transfers which are taking place at that peer.
3. Check the FTP ARRAY and REP ARRAY to verify whether the replicated files are being used currently for file transfers.
4. Get he IP address of those peers and send REPDEL packet to all the peers other than these. On receiving the REPDEL packet the server process deletes the replicated files.
5. When a particular peer being used for file transfer logs out send a REROUTE packet to the peers that are involved in the file transfer with the IP address of the peer where the file has been replicated.
6. A Server socket is already listening at a predefined port.
7. A client socket is established with this predefined port by the peer which receives the REROUTE packet.
8. The requesting peer uses this client socket to send the number of bytes transferred so far and the name of the file to the server process.
9. The server acknowledges with a ready packet and resumes the file transfer.

TESTING

UNIT TESTING

FILE TRANSFER PROTOCOL

First and foremost the protocol for file transfer, which is fundamental to our system, will be tested, verified and validated for correctness and completeness. The client and server programs will be executed on a single system and then will be distributed across different systems and then checked. Initially only one-to-one client/server communication will be established to check whether the file is being transmitted without any errors in parameters like size of file, contents of file, etc. After this step a many-to-one communication where in many clients send requests to a single file server process will be tested.

REPLICATION

After having thoroughly tested the FTP the next step will be to test the protocol design for the replication subsystem. Preliminary tests will be conducted to find out whether replication requests are broadcasted to all logged in peers. The logged in peers should give a positive acknowledgement based on their replication space specified during configuration. Based on the acknowledgement received the currently logged in peer should select the appropriate peer where he wants to replicate his file.

REMOTE COMPUTING

The test for the remote computing subsystem will include a check to find out whether the resource requests are broadcasted to all the logged in peers. The logged in peers should send a positive acknowledgement based on the resources they share within a specified time. The requesting peer should select the source based on the earliest acknowledgement received. Tests should be conducted to verify whether the error status of the execution is properly reported to the requesting peer. If there is no error in the execution then the processed file should be properly returned to the requestor.

LOGOUT

When file transfer is taking place between any two peers, a logging out peer should properly detect such instances and redirect the file transfer to other peers where its files have been replicated before logout. It is tested to make sure that the logging out peer deletes its replicated files across all other peers that have logged in. The peers who have replicated files in the logged out peer should repeat their replication cycle.

INTEGRATION TESTING

The replication and the FTP modules were integrated to form the login subsystem. Tests will be conducted to ensure that the details of all the logged in peers are obtained and maintained in an appropriate data structure. The details in the data structure will be validated and verified. Checks are made to ensure that the replication module uses these details along with FTP to replicate the files in the network utilizing the storage space efficiently and effectively. The application services like File Searching and Remote Computing will be integrated with the backbone and will be tested with various scenarios and test cases. The logout module should be able to access details regarding where the files have been replicated and should possess the capability of querying the FTP modules for activity.

ACCEPTANCE TESTING

The integrated system will be deployed over an existing Local Area Network (LAN). The above tests will be conducted for the integrated system to check whether all the functionalities, which were specified in the SRS, are satisfied. The integrated system is tested for robustness across varying conditions of network traffic and load. The front end is tested for acceptance across users with varying degrees of skill.

OPTIMIZATION AND FUTURE ENHANCEMENTS

- In order to improve the efficiency of file transfer very frequently used files can be replicated in more than one peer. This may increase redundancy but will speed up access.

- The protocol for replication may be improved by incorporating changes such that replication is not localized among a select group of peers but instead files are distributed across the entire intranet increasing storage efficiency.

- This system can be extended to incorporate internetworking. In such a case every network will elect a super peer, which plays a role of a mediator in communicating with super peers of other networks and peers of the same network. The protocol complexity of the backbone will increase but a complete internetworking environment will be available to the user.

- Suppose a peer wants to have a one-to-one communication between a peer who is not present in his network then an option to directly specify the IP address of the peer can be provided using which the above can be achieved.

- The file and resource searching facility, which enables a peer to search for a specific resource across the network, can be made more user-interactive wherein a user has a variety of options to select from.

- In the current system, the remote computing feature is largely dependent on the command line specified and cannot be changed by the user. This means that the system is not extensible for all kinds of computing resources and hence puts a restriction on the type of resource that can be shared. The system can be made more dynamic by allowing the user to specify the command line structure for each resource he shares.

- By making a few changes to the existing system it can be made extensible so as incorporate portability and can be used on a variety of operating systems with the functionalities remaining intact.

CONCLUSION

Our system is a prototype model for a replicating peer-to-peer system and can be integrated into a real time environment by incorporating certain features required for real time network operations. On completion of this project we have tried to develop such a prototype system, which we hope will be a precursor for more advanced P2P systems. Our major goal of developing a self-organizing and reliable P2P system has been achieved through efficient protocol implementation and we have also successfully added essential services like file sharing cum searching, remote computing, instant messaging and on testing the entire system we obtained encouraging results. We hope that through this project we have taken a significant step towards becoming competent IT professionals and kindle the flame of progress and prosperity.

BIBLIOGRAPHY

1. Andy Oram, "Harnessing the Power of Disruptive Technologies"- O'Reilly Pubn., 2001

2. Gittleman Art, "Internet Programming with JAVA 2 Platform" - PHI Pubn., 1997

3. Merlin & Hughes, "Java Network Programming" - Manning Publications Co.1996

4. David M. Geary, "Graphic Java Mastering the JFC" - Sun Microsystems Press.1999

5. Herbert Schildt & Patrick Naughton, The Complete Reference Java2 TMH Pubn.,2000

6. www.felter.org/wesley.

7. www.peer-to-peerwg.org.

8. www.openp2p.com

9. www.sun.java.com

APPENDICES

SAMPLE CODING

LOGGING IN:

```
import java.net.*;
import java.io.*;
import javax.swing.JTabbedPane;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.StringTokenizer;

public class connect1 extends JPanel implements ActionListener
{
        // declaration
        static public newlog log;
        static public searchclient find;
        static public remoteclient remote;
        static public repclient rep;
        static public reptable rept;
        static public finallogoutclient logoutobj;
        //CHAT
        static String error="noerror";
        public static BufferedReader err;
        Process proc;
        String cmd;

        //REMOTE
        JTextField remotename,remotefile;
        JButton remotesearch,compute,remotebrowse;
        JLabel labelremote,textremote,remotehost;
        JComboBox remotelist;
        String fn1;

        //SEARCH
        JTextField searchname;
        JButton search,transfer;
        JRadioButton desc,sname;
        JList listsearch;
```

```
JLabel labelsearch,textsearch;
DefaultListModel modelsearch;
ButtonGroup d;
JProgressBar progressBar;
//CONNECT
JButton connect;
public static JFrame frame1;

//FTP
static public JList listip;
static public JLabel labelip,file,summa;
static public JButton send,exit,browse;
static public JTextField fname;
static public DefaultListModel listModel;
static public String dir,fn;
Timer timer;
InetAddress ftpip;
String ftpfn;
//LOGOUT
 JButton logout;

public connect1(JFrame frame)
{

        ImageIcon icon = new ImageIcon("images/middle.gif");
        JTabbedPane tabbedPane = new JTabbedPane();
        Component panel1 = makePanel1();
        tabbedPane.addTab("CONNECT", icon, panel1, "get connected");
        tabbedPane.setSelectedIndex(0);
        Component panel2 = makePanel2();
         tabbedPane.addTab("FILE TRANSFER", icon, panel2, "transfer
        files");
        Component panel3 = makePanel3();
        tabbedPane.addTab("FILE SEARCH", icon, panel3, "search for file");
        Component panel4 = makePanel4();
        tabbedPane.addTab("REMOTE COMPUTING", icon, panel4, "remote
        computing");
        Component panel5 = makePanel5();
        tabbedPane.addTab("LOG OUT", icon, panel5, "logging out");
        Component panel6 = makePanel6();
        tabbedPane.addTab("CHATTING", icon, panel6, "chatting facilities");
        setLayout(new GridLayout(1, 1));
        timer.start();
        add(tabbedPane);
}

//CONNECT
```

```
    protected Component makePanel1()
     {
            String text="CONNECT";
            JPanel panel = new JPanel(false);
            JLabel filler = new JLabel(text);
            summa = new JLabel("        ");
            filler.setHorizontalAlignment(JLabel.LEFT);
            connect = new JButton("CONNECT");
            panel.add(connect);
            connect.addActionListener(this);
            return panel;
    }

//FTP
    protected Component makePanel2()
    {
            listModel = new DefaultListModel();
            listip = new JList(listModel);
            listip.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
            listip.setSelectedIndex(0);
            JScrollPane listScrollPane = new JScrollPane(listip);
            send=new JButton("SEND");
            exit=new JButton("EXIT");
            browse=new JButton("BROWSE");
            fname=new JTextField(35);
            labelip=new  JLabel("IP ADDRESS");
            summa = new  JLabel("                ");
             file=new  JLabel("FILE NAME");
            send.addActionListener(new SEND());
            exit.addActionListener(new EXIT());
            browse.addActionListener(new BROWSE());
            JPanel buttonPane = new JPanel();
            buttonPane.add(labelip);
            buttonPane.add(listip);
            buttonPane.add(summa);
            buttonPane.add(file);
            buttonPane.add(fname);
            buttonPane.add(browse);
            buttonPane.add(send);
            buttonPane.add(exit);
            timer = new Timer(1000, new ActionListener() {
            public void actionPerformed(ActionEvent evt)
            {
                    int index;
                    index=listip.getSelectedIndex();
                   update();
                   listip.setSelectedIndex(index);
```

```
    }
        });

        return buttonPane;
    }

//FTP SEARCH
    protected Component makePanel3()
    {
        JPanel panel = new JPanel(false);
        summa = new JLabel("                    ");
        textsearch = new JLabel("SEARCH");
        textsearch.setHorizontalAlignment(JLabel.LEFT);
        searchname = new JTextField(25);
        search = new JButton("SEARCH");
        desc = new JRadioButton("SEARCH BY DESCRIPTION");
        sname = new JRadioButton("SEARCH BY FILE NAME");
        d = new ButtonGroup();
        d.add(desc);
        d.add(sname);
        transfer = new JButton("TRANSFER");
        modelsearch = new DefaultListModel();
        listsearch = new JList(modelsearch);
                listsearch.setSelectionMode(ListSelectionModel.
            SINGLE_SELECTION);
        listsearch.setSelectedIndex(0);
        JScrollPane listScrollPane = new JScrollPane(listsearch);
        panel.add(desc);
        panel.add(sname);
        panel.add(textsearch);
        panel.add(searchname);
        panel.add(summa);
        panel.add(search);
        panel.add(listsearch);
        panel.add(transfer);
        search.addActionListener(new SEARCH());
        transfer.addActionListener(new TRANSFER());
        return panel;
    }

    //REMOTE COMPUTING
    protected Component makePanel4()
    {
        JPanel panel = new JPanel(false);
        textremote = new JLabel("SEARCH");
        textremote.setHorizontalAlignment(JLabel.LEFT);
```

```
        labelremote = new JLabel("FILE NAME");
        labelremote.setHorizontalAlignment(JLabel.LEFT);
        remotehost = new JLabel("host name");
        remotehost.setHorizontalAlignment(JLabel.LEFT);
        remotename = new JTextField(25);
        remotefile = new JTextField(40);
        JLabel summa1 = new JLabel("     ");
        summa = new JLabel("     ");
        remotesearch = new JButton("SEARCH");
        compute = new JButton("COMPUTE");
        remotebrowse=new JButton("BROWSE");
        remotelist=new JComboBox();
        panel.add(textremote);
        panel.add(remotename);
        panel.add(remotelist);
        panel.add(remotesearch);
        panel.add(summa);
        panel.add(summa1);
        panel.add(labelremote);
        panel.add(remotefile);
        panel.add(remotebrowse);
        panel.add(compute);
        remotesearch.addActionListener(new REMOTESEARCH());
        compute.addActionListener(new COMPUTE());
        remotebrowse.addActionListener(new REMOTEBROWSE());
        return panel;
}

//LOGOUT
protected Component makePanel5()
{
        String text="LOGOUT";
        JPanel panel = new JPanel(false);
        JLabel filler = new JLabel(text);
        filler.setHorizontalAlignment(JLabel.LEFT);
        logout = new JButton("LOGOUT");
        panel.add(logout);
         logout.addActionListener(new LOGOUT());
        return panel;
}

//chat
protected Component makePanel6()
 {
        JPanel panel= new JPanel(false);
        JButton startchat= new JButton("START CHAT");
        startchat.addActionListener(new CHAT());
```

```
        panel.add(startchat);
        return panel;
}


static public void update()
{
        try
        {
                InputStream pin=new
                FileInputStream("c:\\docume~1\\99cse64\\desktop\\loggedpeer.txt"
                );
                BufferedReader pbuf=new BufferedReader(new
                InputStreamReader(pin));
                listModel.clear();
                String temp;
                while((temp=pbuf.readLine())!=null)
                {
                temp.trim();
                listModel.addElement(temp);
                }
                pbuf.close();
                pin.close();
                        listip.setSelectionMode(ListSelectionModel.
                SINGLE_SELECTION);

        }
        catch(Exception e)
        {e.printStackTrace();}
}


public void actionPerformed(ActionEvent e)
{
        int nop=log.startlog();
        if(nop==0)
        {
                JOptionPane.showMessageDialog(new Frame(),"NO PEERS IN
                NETWORK","Confirmation",0);
        }
        else
        {
                JOptionPane.showMessageDialog(new Frame(),"CONNECTED
                TO HRPP"+nop+"PEERS LOGGEDIN","Confirmation",0);

                rep.repli(rept);
        }
```

```
//servers execution
try
{
        String cmd1= "c:\\jdk\\bin\\javaw.exe -jar
        z:\\hrpp\\login\\newdest.jar";
        String cmd2= "c:\\jdk\\bin\\javaw.exe -jar
        z:\\hrpp\\login\\finalrepserver.jar";
        String cmd3= "c:\\jdk\\bin\\javaw.exe -jar
        z:\\hrpp\\login\\searchserver.jar";
        String cmd4= "c:\\jdk\\bin\\javaw.exe -jar
        z:\\hrpp\\login\\remoteserver.jar";
        String cmd5= "c:\\jdk\\bin\\javaw.exe -jar
        z:\\hrpp\\login\\chatserever.jar";
        String cmd6= "c:\\jdk\\bin\\javaw.exe -jar
        z:\\hrpp\\login\\Trserver.jar";
        String cmd7= "c:\\jdk\\bin\\javaw.exe -jar
        z:\\hrpp\\login\\finallogoutserver.jar";
        System.out.println("Running"+cmd1);
        System.out.println("Running"+cmd2);
        System.out.println("Running"+cmd3);
        System.out.println("Running"+cmd4);
        System.out.println("Running"+cmd5);
        System.out.println("Running"+cmd6);
        System.out.println("Running"+cmd7);
        Process proc1 = Runtime.getRuntime().exec(cmd1);
        Process proc2 = Runtime.getRuntime().exec(cmd2);
        Process proc3 = Runtime.getRuntime().exec(cmd3);
        Process proc4 = Runtime.getRuntime().exec(cmd4);
        Process proc5 = Runtime.getRuntime().exec(cmd5);
        Process proc6 = Runtime.getRuntime().exec(cmd6);
        Process proc7 = Runtime.getRuntime().exec(cmd7);
}
catch(Exception e1)
{e1.printStackTrace();}
try
{
        proc1.waitFor();
        proc2.waitFor();
        proc3.waitFor();
        proc4.waitFor();
        proc5.waitFor();
        proc6.waitFor();
        proc7.waitFor();
}
catch (InterruptedException e)
{
        System.err.println("process was interrupted");
```

```
        }
        if (proc1.exitValue() != 0)
{
                error="error";
                System.err.println("exit value was non-zero"+error);
         }
        if (proc2.exitValue() != 0)
        {
                error="error";
                System.err.println("exit value was non-zero"+error);
        }
        if (proc3.exitValue() != 0)
        {
                error="error";
                System.err.println("exit value was non-zero"+error);
        }
        if (proc4.exitValue() != 0)
        {
                error="error";
                System.err.println("exit value was non-zero"+error);
        }
        if (proc5.exitValue() != 0)
        {
                error="error";
                System.err.println("exit value was non-zero"+error);
        }
        if (proc6.exitValue() != 0)
        {
                error="error";
                System.err.println("exit value was non-zero"+error);
        }
        if (proc7.exitValue() != 0)
        {
                error="error";
                System.err.println("exit value was non-zero"+error);
        }

}

class LOGOUT implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                try
                {

                        logoutobj.startlogout(ftpip,ftpfn);
```

```
                }
                catch(Exception e1){}
        }
}


class CHAT implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                try
                {
                        cmd="c:\\jdk\\bin\\javaw.exe -jar
                        z:\\hrpp\\chat\\chatclient.jar";
                        Process proc = Runtime.getRuntime().exec(cmd);
                        err = new BufferedReader(new
                        InputStreamReader(proc.getErrorStream()));
                }
                catch(Exception e2){}


                try
                {
                        proc.waitFor();
                }
                catch (Exception e1)
                {
                        System.err.println("process was interrupted");
                }



                if (proc.exitValue() != 0)
                {
                        error="error";
                        System.err.println("exit value was non-zero"+error);
                }

        }
}

class REMOTESEARCH implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                try
                {
```

```
                    String putlist[]=remote.rclient(remotename.getText());
                    for(int h=0;h<putlist.length;h++)
                    {
                        remotelist.addItem(putlist[h]);

                    }

                    remotelist.setSelectedIndex(0);
                    remotelist.setVisible(true);

                }
                catch(Exception e1)
                {
                    System.out.println("remotesearh");
                }
        }
}

class REMOTEBROWSE implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                try
                {
                Frame f1=new Frame("Remote");
                FileDialog fd1=new FileDialog(f1,"Remote FD");
                fd1.show();
                fn1=fd1.getFile().toString();
                String fullpath=fd1.getDirectory().toString()+fn1;
                remotefile.setText(fullpath);

                }
                catch(Exception e1)
                {
                    System.out.println("remote browse");
                }
        }
}

class COMPUTE implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {

                try
                {
                    String selitem=remotelist.getSelectedItem().toString();
```

```
                    StringTokenizer str_tok2=new
                    StringTokenizer(selitem,"?");
                    str_tok2.nextToken();
                    StringTokenizer str_tok1=new
                    StringTokenizer(str_tok2.nextToken(),"/");
                    str_tok1.nextToken();
                    String ip=str_tok1.nextToken();
                    InetAddress ipaddr=InetAddress.getByName(ip);
                    remote.rcompute(remotefile.getText(),ipaddr,fn1);
                    if(remote.err_str.equals("error"))
                    {
                    JOptionPane.showMessageDialog(new Frame(),"Error in
                    file please verify","status ",0);
                    }
                    else
                    {
                    JOptionPane.showMessageDialog(new Frame(),"Computed
                    file ","status",0);
                    }
                }
                catch(Exception e6){e6.printStackTrace();}

        }
    }
class SEARCH implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            String send=new String();
            if(desc.isEnabled())
                send="fna?"+searchname.getText();
            if(sname.isEnabled())
                send="des?"+searchname.getText();
            String[] list=new String[25];
            list=find.start_searchclient(send);
            int i;
            for(i=0;i<list.length;i++)
                modelsearch.addElement(list[i]);

        }
    }

    class TRANSFER implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            try
```

```
                {
                        String ipad=listsearch.getSelectedValue().toString();
                        StringTokenizer str_tok=new StringTokenizer(ipad,"?");
                        str_tok.nextToken();
                        String ip=str_tok.nextToken();
                        String toserver_fname=str_tok.nextToken();
                        byte buf[] =toserver_fname.getBytes();
                        StringTokenizer str_add=new StringTokenizer(ip,"/");
                        str_add.nextToken();
                        String sendadd = str_add.nextToken();
                        InetAddress ipaddr = InetAddress.getByName(sendadd);
                        DatagramSocket search = new DatagramSocket();
                        DatagramPacket filepkt = new
                        DatagramPacket(buf,buf.length,ipaddr,9191);
                        search.send(filepkt);
                        FileDialog save=new FileDialog(new Frame(),"SAVE
                        AS",FileDialog.SAVE);
                        save.setVisible(true);
                        String savepath= (save.getDirectory() +
                        save.getFile()).toString();
                        System.out.println("start client:");
                        DatagramPacket filepack = new
                        DatagramPacket(buf,buf.length,ipaddr,9292);
                        search.send(filepack);
                        search_ftpclient startftp=new search_ftpclient();
                        startftp.transferfile(savepath);
                        modelsearch.clear();

                }
                catch(Exception e1)
                {
                        System.out.println("Exception in connect");
                }
        }
}

class SEND implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                try
                {
                        ftpbackend ftp=new ftpbackend();
                        System.out.println("send
                        clicked"+listip.getSelectedIndex());
                        InetAddress ip = InetAddress.getByName(listip
                        .getSelectedValue().toString());
```

```
                    File f1=new File(dir+fn);
                    System.out.println("CALLED FTP BACKEND");
                    ftpip=ip;
                    ftpfn=dir+fn;
                    ftp.filetrans(ip,dir.fn,f1.length());
                    ftpip=null;
                    ftpfn=null;
                }
                catch(Exception e1)
                {System.out.println("Exception in send");}
            }
        }


class BROWSE implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            Frame f1=new Frame("Client");
            FileDialog fd1=new FileDialog(f1,"Client FD");
            fd1.show();
            dir=fd1.getDirectory().toString();
            fn=fd1.getFile().toString();
            fname.setText(dir+fn);
        }
    }

    class EXIT implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            if((JOptionPane.showConfirmDialog(new Frame(),"Are You Sure
            to close the Session?"))==JOptionPane.YES_OPTION)
            System.exit(0);
        }
    }



public void remote()

 {
    //declaration
    log = new newlog();
    find = new searchclient();
    remote = new remoteclient();
    rep=new repclient();
```

```
        rept=new reptable();
        logoutobj=new finallogoutclient();
       frame1 = new JFrame("CONNECT");
        frame1.addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e) {System.exit(0);}
     });
      frame1.getContentPane().add(new connect1(frame1),
      BorderLayout.CENTER);
       frame1.setSize(700, 200);
       frame1.setResizable(false);
       frame1.setLocation(90,150);
       frame1.setVisible(true);
   }
}
```

FILE TRANSFER :

```
import java.net.*;
import java.io.*;
import java.util.StringTokenizer;
public class ftpserver
        {
                public static void main(String[] args)
                {
                        String s;
                        byte c;
                        int n=0;
                        try
                        {
                                ServerSocket server=new ServerSocket(50000);
                                Socket client = server.accept();
                                System.out.println("connected on port"+50000);
                                BufferedReader fromclient =new BufferedReader(new
                                InputStreamReader(client.getInputStream()));
                                s=fromclient.readLine();
```

```
                    long size=Long.valueOf(fromclient.
                    readLine()).longValue();
                    System.out.println("ftpserver"+s+"" +size);
                    String dir="z:\\hrpp\\junk1\\";
                    File f1=new File(dir+s);
                    System.out.println("bye");
                    FileOutputStream intofil = new FileOutputStream(dir+s);
                    System.out.println("hai");
                    while(n<size)
                    {
                            c=(byte)Integer.parseInt(fromclient.readLine());
                            intofil.write(c);
                            System.out.print(n+"bytes transferred\r");
                            n++;
                    }
                    server.close();
                    intofil.close();
                    client.close();
            }
            catch(Exception e)
            {
            e.printStackTrace();
            }
    }}
```

SEARCH SERVER:

```
import java.net.*;
import java.io.*;
import java.util.StringTokenizer;

public class searchserver
{
        public static void main(String args[])
        {
        try
        {
                byte name[] =new byte[100];
                DatagramSocket search;
                DatagramPacket filepkt;
         while(true)
```

```
                {
                        System.out.println("waiting for requests:");
                        search = new DatagramSocket(25000);
                        filepkt = new DatagramPacket(name,name.length);
                        search.receive(filepkt);
                        new searchthread(filepkt);
                        search.close();
                }
        }
        catch(Exception e){}
        }
}

class searchthread extends Thread

{
        DatagramPacket filepkt;
        DatagramSocket recv_file;
        public searchthread(DatagramPacket filepkt)
        {
                this.filepkt=filepkt;
                start();
        }
        public void run()
        {
        try
        {
                String msg=new String(filepkt.getData());
                 msg=msg.trim();

                System.out.println("msg="+msg);
                StringTokenizer str1=new StringTokenizer(msg,"?");
                String stype=str1.nextToken();
                String file=str1.nextToken();
                System.out.println("type="+stype +file+"sa");
                String profile="z:\\hrpp\\profile\\profile1.txt";
                File fil=new File(profile);
                FileInputStream fstream=new FileInputStream(fil);
                BufferedReader read=new BufferedReader(new
                InputStreamReader(fstream));
                System.out.println(fstream);
                String[] filename=new String[10];
                String[] filepath=new String[10];
                int i=0,ok=0;
                if(stype.equals("des"))
        {
                        System.out.println("inside");
```

```
                String line;
                while((line=read.readLine())!=null)
                {

                        System.out.println("line="+line);
                        StringTokenizer str=new StringTokenizer(line,"?");
                        str.nextToken();
                        String fpath=str.nextToken();
                        str.nextToken();
                        String fname=str.nextToken();
                        str.nextToken();
                        String desc=str.nextToken();
                        System.out.println(desc);
                        if(desc.equals(file))
                        {
                                System.out.println("inside if..");
                                filename[i]=fname;
                                filepath[i]=fpath;
                                System.out.println("fname="+fname);
                                i++;

                        }

                }
                System.out.println("outside while:"+i);
        }//endif
else
{
        System.out.println("inside");
        String line;
        while((line=read.readLine())!=null)
        {
                System.out.println(i+line);
                StringTokenizer str=new StringTokenizer(line,"?");
                str.nextToken();
                String fpath=str.nextToken();
                str.nextToken();
                String fname=str.nextToken();
                if(fname.equals(file))
                {
                        System.out.println("fname="+fname);
                        filename[i]=fname;
                        filepath[i]=fpath;
                        i++;

                }
        }
```

```
        }
        for(int j=0;j<i;j++)
        {
                System.out.println("Sendin resuls.."+filename[j]);
                Socket send_searchclient=new Socket(filepkt.getAddress(),30001);
                PrintWriter to_searchclient=new
                PrintWriter(send_searchclient.getOutputStream(),true);
                File f=new File(filepath[j]);
                to_searchclient.println(filename[j]+"?"+f.length());
                send_searchclient.close();

        }
        byte buf[] = new byte[100];
        recv_file = new DatagramSocket(9191);
        recv_file.setSoTimeout(20000);
        DatagramPacket recvpkt = new DatagramPacket(buf,buf.length);
        recv_file.receive(recvpkt);
        String rec = new String(recvpkt.getData());
        rec = rec.trim();
        System.out.println("received fromclient fname:  "+rec);
        String fullpath=null;
        for(int z=0;z<i;z++)
        {
                if(rec.equals(filename[z]))
                {
                 fullpath=filepath[z];

                }
        } .

        System.out.println("full path=  "+fullpath);
        DatagramSocket recv = new DatagramSocket(9292);
        DatagramPacket recpkt = new DatagramPacket(buf,buf.length);
        recv.receive(recpkt);
        System.out.println("Starting server:");
        search_ftpserver ftpserver=new search_ftpserver();
        ftpserver.sendfile(fullpath,recvpkt.getAddress());
        recv_file.close();
        recv.close();
}
catch(Exception e)
{
        e.printStackTrace();
}
       recv_file.close();
}
```

```
}
```

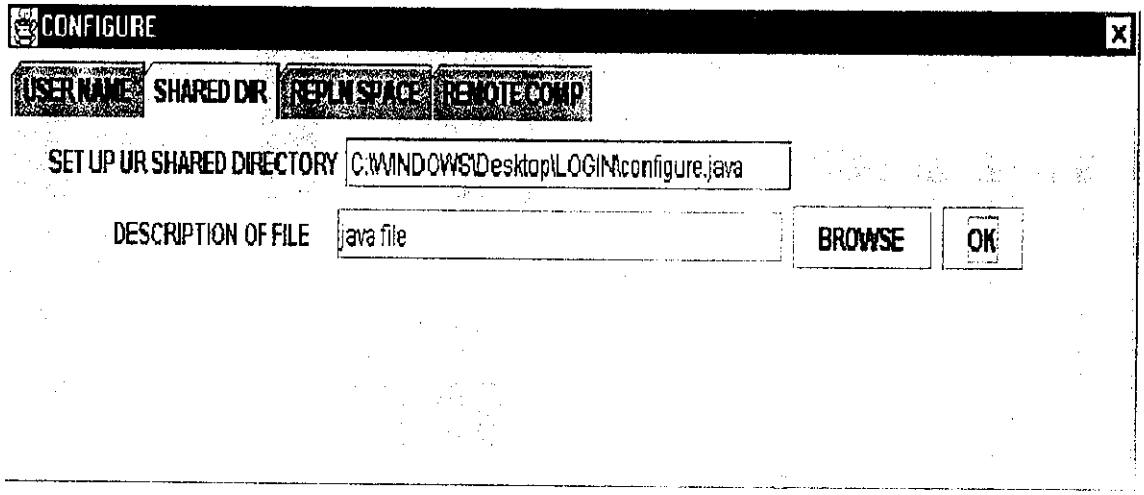
REMOTE SERVER:

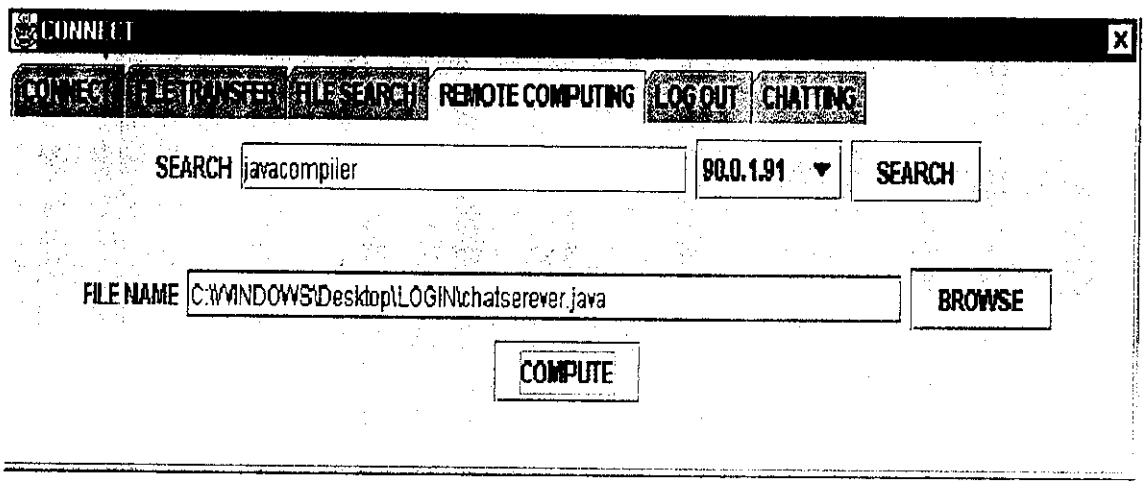
```
import java.net.*;
import java.io.*;
import java.util.StringTokenizer;
public class remoteserver
{
        static String check;
        public static void main(String args[])
        {
                int SEND=7777;
                try
                        {
                        byte buf[] = new byte[255];
                        int len;
                        DatagramSocket resrecv =  new DatagramSocket(6464);
                        DatagramSocket resrec =  new DatagramSocket(6465);
                        DatagramPacket res_recv = new DatagramPacket(buf,buf.length);

                        DatagramSocket resend =  new DatagramSocket();
                        resrecv.receive(res_recv);
                        InetAddress peer=res_recv.getAddress();
                        resrecv.close();
                        String str=new String(res_recv.getData());
                        String file=str.trim();
                        System.out.println("String is:"+file);
                        String profile="z:\\hrpp\\profile\\profile1.txt";
                        File fil=new File(profile);
                        FileInputStream fstream=new FileInputStream(fil);
                        BufferedReader read=new BufferedReader(new (fstream));
                        String[] filepath=new String[10];
                        String[] filedesc=new String[10];
                        String line;
                        int i=0,ok=0;
                        while((line=read.readLine())!=null)
                        {
                                System.out.println("line="+line);
                                StringTokenizer str1=new StringTokenizer(line,"?");
                                str1.nextToken();
                                String fpath=str1.nextToken();
                                for(int r=0;r<3;r++)
                                {
                                        str1.nextToken();
                                }
```

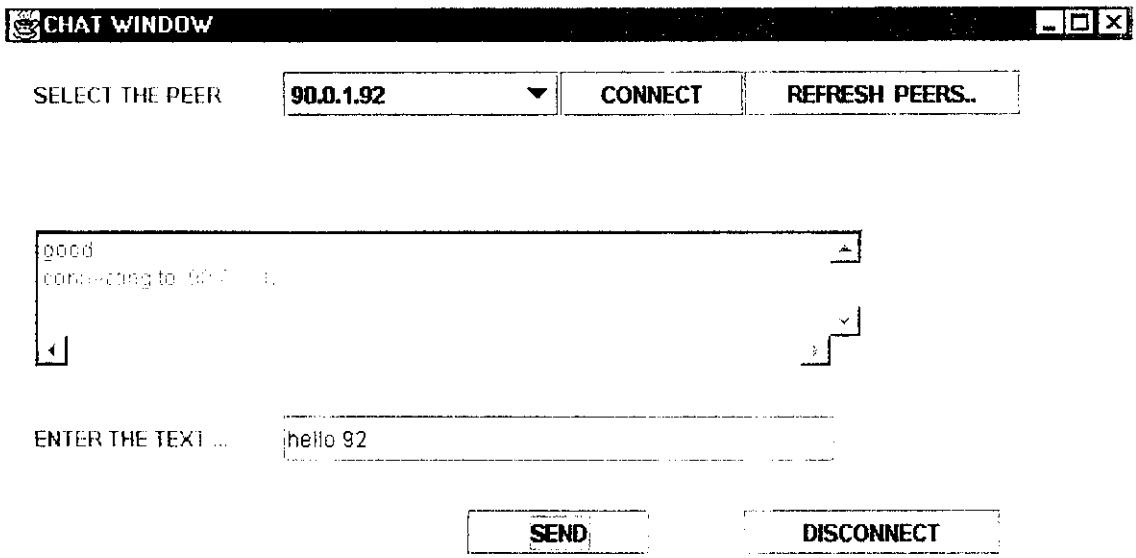
```
                String desc=str1.nextToken();
                System.out.println(desc);
                if(desc.equals(file))
                {
                        System.out.println("inside if..");
                        filedesc[i]=desc;
                        filepath[i]=fpath;
                        System.out.println("fdesc="+desc);
                        i++;

                }

        }
        System.out.println("Sendin resuls.."+filedesc[i-1]);
        Socket send_remoteclient=new
        Socket(res_recv.getAddress(),6565);
        PrintWriter to_remoteclient=new
        PrintWriter(send_remoteclient.getOutputStream(),true);
        to_remoteclient.println(filedesc[i-1]);
        send_remoteclient.close();
        byte rbuf[]=new byte[200];
        String sendfile=null;
        res_recv = new DatagramPacket(rbuf,rbuf.length);
        resrec.receive(res_recv);
        resrec.close();
        String recstr=new String(res_recv.getData());
        String recv_str=recstr.trim();
        System.out.println("recveieed.."+recv_str);
        String str1="send";
        if(recv_str.equals("OK"))
        {
                byte sbuf[]=str1.getBytes();
                len=sbuf.length;
                DatagramPacket res_send = new
                DatagramPacket(sbuf,len,peer,7777);
                resend.send(res_send);
                resend.close();
                ftpserver1 recv=new ftpserver1(peer);
                String fname=recv.filename;
                StringTokenizer tok=new StringTokenizer(fname,".");
                String s=tok.nextToken();
                String extn=".class";
                sendfile=s+extn;
                Execute cmd=new Execute();
                String commd=filepath[i-1]+" "+"Z:\\hrpp\\recv\\"+fname;
                System.out.println("Exceuting.."+commd);
                check=cmd.runCommand(commd);
```

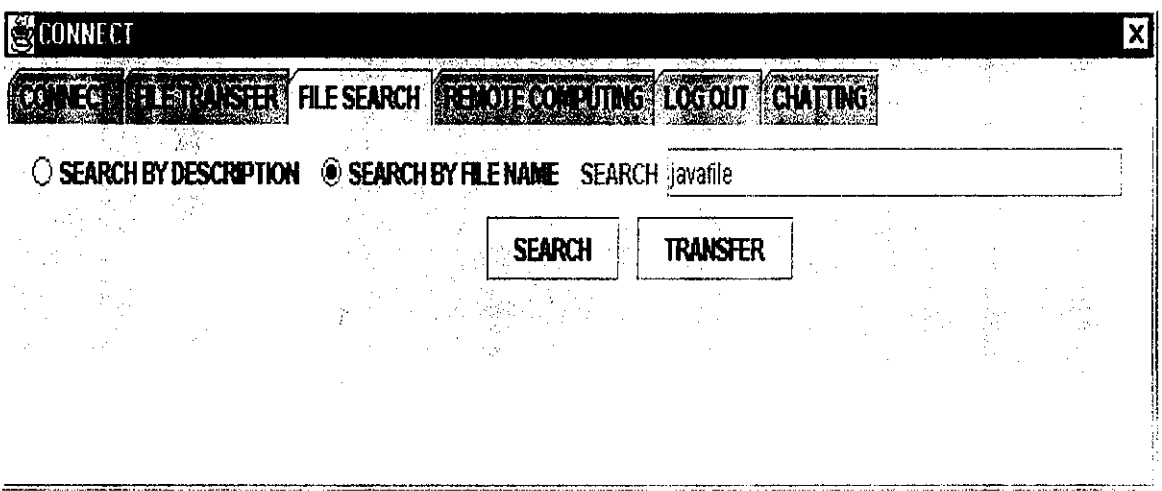
```
                    System.out.println("Executed:");

                }

                DatagramSocket err_sock =  new DatagramSocket();
                System.out.println("Sendin error status:");
                System.out.println("check="+check);

                if(check.equals("error"))
                {
                        System.out.println("Error:");
                        byte buf1[]=check.getBytes();
                        DatagramPacket err_pkt = new
                        DatagramPacket(buf1,buf1.length,peer,2222);
                        err_sock.send(err_pkt);
                        err_sock.close();
                        System.exit(0);

                }
                else
                {
                        check="noerror";
                        byte buf2[]=check.getBytes();
                        DatagramPacket err_pkt = new
                        DatagramPacket(buf2,buf2.length,peer,2222);
                        err_sock.send(err_pkt);
                        err_sock.close();

                }
                System.out.println("Sendin file"+sendfile);
                ftpclient1 send=new ftpclient1(peer,sendfile);
        }
        catch(Exception e)
        {
                e.printStackTrace();
                System.out.print("Exception");
        }

    }

}
```

USER INTERFACE






CHAT WINDOW
SELECT THE PEER
90.0.1.92
CONNECT
REFRESH PEERS..
good
ENTER THE TEXT ...
hello 92
SEND
DISCONNECT


CONNECT
CONNECT
FILE TRANSFER
FILE SEARCH
REMOTE COMPUTING
LOG OUT
CHATTING
SEARCH BY DESCRIPTION
SEARCH BY FILE NAME
SEARCH
javafile
SEARCH
TRANSFER