Reliable File Transfer on the Internet using Distributed File Transfer (DFT)

by

Xin Fang

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba, Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

Reliable File Transfer on the Internet using

Distributed File Transfer (DFT)

BY

Xin Fang

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science

I hereby declare that I am the sole author of his thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

ABSTRACT

Transferring files over the Internet is extremely common, hence the protocol, FTP. However, the functionality, and reliability of FTP relies on lower layer protocols. Improved performance mechanisms have yet to be deployed that combine the ubiquity of FTP, high reliability, and fault tolerance. This thesis describes the design and implementation of such a mechanism called Distributed File Transfer (DFT). DFT establishes multiple FTP sessions with multiple FTP sites transferring different segments of a file simultaneously. It switches to other servers if some server becomes overloaded or unavailable. A Load Distributing Server (LDS) in DFT collects server information and distributes load among multiple servers. Experiments presented show that DFT is both reliable and efficient.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all the people who have contributed towards this thesis.

I would first like to thank my advisor, Dr. Robert D. McLeod, for his guidance and encouragement through my academic years as well as his contributions and help with this thesis. In addition, I would like to express my appreciation for his kind and generous personality.

I would also like to thank Dr. Muthucumaru Maheswaran and Dr. David C Blight, for taking the time to be on my thesis committee for reading my thesis and for their efforts during my studies.

Furthermore, I would like to thank my husband, Jingshao Chen, and my parents, for their constant encouragement and support.

TABLE OF CONTENTS

Approval Form i

Abstract iii

Acknowledgements iv

Table of Contents v

List of Figures x

List of Tables xii

Chapter 1 Introduction 1

1.1 Distributed Processing 2

1.2 The Internet 4

1.3 File Transfer Protocol (FTP) 5

1.4 Motivation 7

1.4.1 Statement of Problems 7

1.4.2 The Objective of DFT 9

Summary 11

Chapter 2 Related Work 12

2.1 FTP 12

2.2 Checkpoint Resume Applications 16

2.3 Round-Robin DNS 17

2.4 Internet Load Balance Solutions 18

2.5 Comparison of DFT and Other Solutions 19

Summary 19

Chapter 3 Distributed File Transfer Architecture 21

3.1 Distribution Architecture of DFT 21

3.1.1 DFT Client 23

3.1.2 Load-Distributing Server (LDS) 24

3.1.3 File servers 26

3.2 DFT Two-layer Connection Model 27

3.2.1 Control-layer Connection 30

3.2.2 Data-layer Connection 32

3.3 DFT Working Procedure 33

3.3.1 Target File Query and Searching 33

3.3.2 Multi-connection File Downloading 35

3.4 Fault Tolerance 37

3.4.1 Layered Failure Model 38

3.4.2 Failure Detection 40

3.4.3 Failure Recovery 43

Summary 44

Chapter 4 DFT Implementation 46

4.1 Implementation Tools 46

4.1.1 Java 46

4.1.2 Java Foundation Classes (JFC) 49

4.2 DFT client's implementation 50

4.2.1 DFT Client's Function Block 50

4.2.2 Major Classes in DFT Client 64

4.3 LDS implementation 66

4.3.1 LDS Function Blocks 66

4.3.2 Major Classes in LDS 69

Summary 70

Chapter 5 Experiments and Data Analysis 71

5.1 Experiment Design 71

5.1.1 Experiment Goals 72

5.1.2 Experiment Environment 74

5.2 Reliability Test 75

5.2.1 Server Failure Recovery 75

5.2.2 Hardware Failure Recovery 77

5.2.3 DFT Client Failure Recovery 77

5.3 Performance Test 77

Summary 81

Chapter 6 Conclusions 83

References 85

Appendix A Numeric Order List of FTP Reply Codes 88

LIST OF FIGURES

Figure 1 the Number of Hosts on the Internet from 1995 to 2000 [ISC2000] 5

Figure 2 Typical DFT System Layout 22

Figure 3 DFT Client and DFT File Servers 27

Figure 4 FTP Transportation Model 28

Figure 5 DFT Logical Transportation Model 29

Figure 6 DFT Control-level Connection and Data-level Connection 30

Figure 7 DFT Two-layer Model and Their Functions 32

Figure 8 Queries and Reply between a DFT Client and an LDS 34

Figure 9 Multiple File Segments Downloading 37

Figure 10 OSI Reference Model and Three-layer Failure Model 38

Figure 11 Job Switching when Failure Occurs in DFT 44

Figure 12 Function Blocks of a DFT Client and the Information Flow between Them 51

Figure 13 Overall Download Procedure of a DFT Client 54

Figure 14 Working Procedure of a Controller 56

Figure 15 Working Procedure of a Download Thread 59

Figure 16 Working Procedure of the Timer 62

Figure 17 Communication between a DFT Client and an LDS 67

Figure 18 Downloading Speed of One Server 78

Figure 19 Downloading Speeds of Two Servers 79

Figure 20 Downloading Speeds of Three Servers 79

Figure 21 Downloading Speeds of Four Servers 80

Figure 22 Throughput of DFT with Different Number of Servers 81

LIST OF TABLES

Table 1 Meanings of Finish-flag and Stall-flag combination 53

Table 2 Explanations of DFT client's overall behavior flow chart 56

Table 3 Explanations of DFT client's download thread controller flow chart 58

Table 4 Explanations of DFT client's download thread flow chart 61

Table 5 Explanations of DFT client's timer flow chart 64

Table 6 DFT servers in experiment 75

Table 7 Failure/recovery time when a server fails 76

Table 8 Failure/recovery time when a server's link fails 77

Table 9 Downloading rate with different number of servers 80

Chapter 1
Introduction

This thesis presents a reliable file transfer mechanism called Distributed File Transfer (DFT). The purpose of DFT is to increase the reliability of transferring files over the Internet.

DFT is a distributed system. It simultaneously transfers a target file from multiple file servers. DFT establishes multiple connections between the DFT client and file servers. All connections transfer different segments of a target file at the same time. By distributing the transfer tasks among multiple distributed servers, DFT provides a reliable and efficient mechanism of file transferring.

DFT is designed to meet the requirements of today's computing environment. It works on a wide geographic distributed network and heterogeneous systems. It uses the most

common and widespread protocol, TCP/IP. Thus, any nodes that have TCP/IP implementation can run DFT. Moreover, the hardware independent language Java was chosen as the programming language to implement DFT. With these two features, DFT can work on most computers throughout the largest distributed environment, the Internet.

1.1 Distributed Processing

Distributed computing environments have been widely adopted to increase a system's reliability and performance. A distributed system normally contains: nodes or hosts and the network itself. Workload is distributed among computing nodes and nodes are connected by the network. The network is responsible for the transfer of information between nodes. The information includes control information, distributed code and distributed data.

There are two kinds of distribution, method distribution and data distribution. Method distribution makes each node do a different task. In method distribution, a large computing task is divided into many small tasks and the nodes will process the small tasks simultaneously. Data distribution distributes replicas of the data or part of the data to different nodes. Thus, if some nodes fail, the data is still available to users. In general, method distribution increases the performance of a system and data distribution increases the reliability.

To achieve reliability, the nodes that contain the data should be widely spread apart. The degree of reliability determines the degree of decentralization. Two servers can be mounted in different racks in the same room. This configuration can endure one server failure but cannot tolerant a power failure of the building. Or we can put servers in different buildings. But this cannot tolerate an earthquake in that area. Sometimes, to avoid a large-scale disaster, the nodes would be distributed across the country or continent.

However, wide scale distribution brings another problem. The more widely the nodes are separated, the larger the network is. The larger the network is, the more complex it is, too. The construction of such a reliable, fast and large network will be very costly. Moreover, this network should be multi-purpose in addition to that particular distributing application.

Therefore, a very large network is needed to increase the reliability and that network should carry the information of the distributed system, as well as other information for different applications. Fortunately, we now have such a network at hand. Moreover, we can use the network with nearly zero cost. It is the Internet.

1.2 The Internet

The Internet was born in 1969. It was conceived by the Advanced Research Projects Agency (ARPA) of the U.S. government and was first known as the ARPANet. The original aim was to create a network that would allow users of a research computer at one university to be able to communicate with research computers at other universities. When it was designed, the idea of reliability also was weaved into the structure of Internet. The Internet is designed as a packet switched peer-to-peer communication system. Therefore, if one router fails, packets can be rerouted through other routers to their destinations. Even if part of the Internet is destroyed, the remaining part can still work.

From the day it was formulated, the Internet has been growing at an exponential rate [Paxson1994]. Figure 1 shows the number of hosts connected to the Internet. According to the research of Paxon, the traffic in the Internet is also growing at an exponential rate. With this fast growth, geographic distribution becomes increasingly cost effective. Many new applications take advantage of global connectivity and the low communication costs of the Internet. A VPN (virtual private network) [Ryan1999] is one example that uses the Internet as a communication network to construct a private network just as a workgroup on a LAN. Other applications such as Web switching [APa] [APb] take full advantage of the redundancy of the Internet to provide quick and efficient service to users.

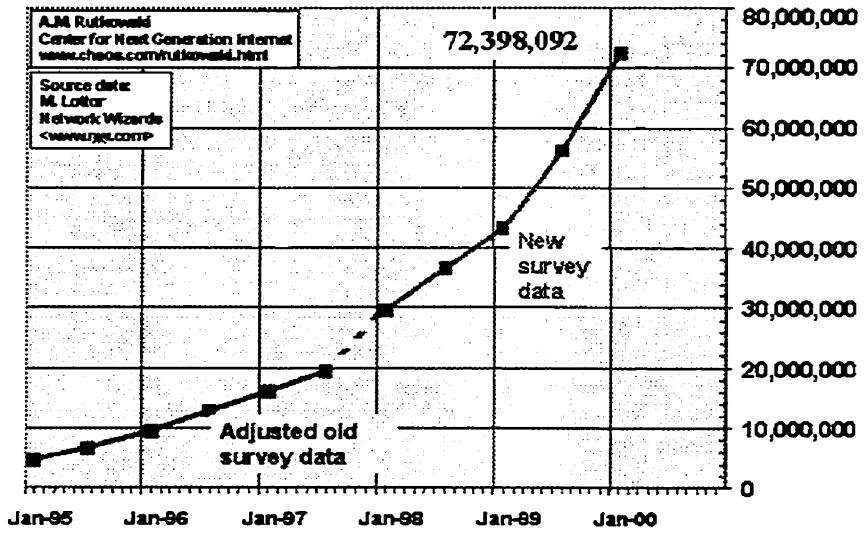

Figure 1 the Number of Hosts on the Internet from 1995 to 2000 [ISC2000]

1.3 File Transfer Protocol (FTP)

Among all the applications on the Internet, file transfer is a very basic but very useful application. FTP is the application that transfers files between FTP clients and FTP servers on the Internet. It has been used since 1971. The widely accepted standard of FTP is defined in RFC 959, 1985 [PR1985]. It specified the application that allows a user to retrieve files from a remote FTP server. FTP relies on TCP. The reliability of TCP provides reliability of FTP itself. It has many features that makes file transfer among heterogeneous system possible. Recent RFCs have developed FTP's security features [HL1997] and FTP over IPv6 [AOM1998].

Today, FTP is the second most widely used application on the Internet in terms of traffic [ISC2000]. Computers that connect to the Internet need to transfer files between each other from time to time. Nearly every software company has their own FTP site for their customers to download the latest products or patches. Many famous web sites such as "www.download.com" and "www.tucows.com" provide a service that collects popular download files over the Internet. Users access their sites and download the file that interests them. Some search engines also have dedicated FTP search engines that can search through all the mirror sites of the Internet for a file. One example is Lycos's search site [Lycos] that can search for on FTP file name.

Because the demand for downloading a file is throughout the Internet, FTP server mirroring is widely implemented. Many FTP sites have many mirror sites distributed around the world. The mirror sites contain the same copies of files as on the original server. Therefore, a user in Winnipeg can download a file from the mirror site that is located in Winnipeg while a user in China can download the same copy of the file from the mirror site in China. This approach generally can keep the traffic local and achieves better performance.

However, if an FTP site has too many user connections, the speed of the site will be slow. Most FTP servers have limits on the maximum number of concurrent users. New users

have to wait until the server has an idle connection. Alternatively, the user can try to find another mirror site or another FTP server that contains the same file and download the file from there. This procedure does not happen automatically, users have to wait or switch servers manually.

1.4 Motivation

1.4.1 Statement of Problems

FTP is widely used and easy to implement. However, it has problems with reliability, performance and scalability. The problems are mostly caused by the growth of the number of users, traffic, and the diversity of the Internet itself. Several problems with traditional FTP include:

1. The load can easily exceed the capacity of the server. Under this circumstance, the server will refuse new connections in order to maintain performance.

2. While some servers are quite busy, other mirror sites may be quite idle in contrast. Users tend to download files from the servers with which they are familiar. Therefore, although there are many mirror sites around the world, the load may not be equally distributed among them.

3. When a server is brought down for upgrade or maintenance, all users will lose their connections with the server and the FTP sessions will be interrupted.

4. New FTP servers need a lot of advertisement to make them known to users.

The reliability problems are apparent when a user is downloading files from a server. The reasons that cause the file transferring process to be seriously degraded or broken are:

1. The link (either physical or logical link) between the client and server is congested or broken.

2. The server is not available due to too many simultaneous users or the system is down.

Looking into the problems that FTP is facing, one notes that poor traffic allocation is a key component of the problem. When a server is too busy, if there is a mechanism that redirects a new user to an idle server, the server unavailability problem will be solved. When a server or the link to the server is down, if there is mechanism that transparently switches the user to another mirror site, there will be a reduction in the number of problems associated with hard faults or outages.

1.4.2 The Objective of DFT

Distributed File Transfer (DFT) is designed to solve these problems. DFT can allocate server resources according to their availability and utilization. Therefore, we obtain reliability, scalability and efficiency by making better use of all file servers in the Internet. This approach makes all FTP servers that contain the same files a "cluster" and distributes load among them.

The design concept of DFT is briefly described as follows:

1. In order to switch among servers, a client connects to multiple file servers to download file components simultaneously. This approach improves the performance of file transfer in general.

2. The client monitors all connections to file servers to detect failure and congestion problems. If some connection is broken, the DFT client should have sufficient intelligence or means to decide which server it should switch to and switch to that server immediately.

3. There should be a Load Distributing Server (LDS). The LDS knows all file servers so that it can provide a server list to a client. Thus, the LDS will take care of the load

distribution among all servers and the client will focus on file transfer among a small set of servers.

With the design concept in mind, the following table gives the main goals that DFT will achieve:

Reliability File transfer is not interrupted even when servers or links are down. Servers can be geographically separated among many sites. DFT should offer anti-disaster level reliability.

Efficiency A client downloads multiple segments simultaneously from a carefully chosen group of servers. File transfer load is dynamically distributed to achieve best performance.

Scalability A new server can be added at any time and anywhere. Load can be distributed to the new server as long as it registers itself with LDS.

Compatibility File transfer is based on the most widely adopted protocol, FTP. This feature makes our system easy to implement and compatible with the

existing Internet.

Transparency The failure recovery process should be transparent to user.

Summary

This section gave a brief background on distributing computing, the Internet and File Transfer Protocol (FTP). Based on that, the problems of current file transferring on the Internet was presented. One of the reasons for file transfer problems is poor load allocation. Therefore, a distributed file transfer mechanism was presented. The last part of this section provided the concepts and the objectives of the DFT methodology.

Chapter 2
Related Work

This section introduces some related work on the reliability of file transferring and load distribution topics. Section 2.1 introduces the very basic protocol, FTP. Section 2.2 introduces applications that take advantages of the checkpoint resume property of FTP. Section 2.3 introduces the round-robin DNS mechanism to distribute load among hosts and section 2.4 introduces the solutions that perform load balancing based on Internet traffic.

2.1 FTP

FTP sessions maintain two connections between an FTP client and an FTP server. One connection is control connection and the other is data connection. The control connection is connected to the well know TCP port 21 on the server end. The data connection port is generally connected to port 20 on the server end. Files are transferred only via the data

connection. The control connection is used for the transfer of commands. FTP commands and responses are transferred in the same manner and format as Telnet.

In FTP, the packet level error detection and recovery is carried out by TCP. However, at the application level, FTP has the checkpoint restart mechanism that user can resume downloading a file from a given checkpoint. The checkpoint restart procedure is defined in block and compress transfer modes. Most FTP servers implement the checkpoint restart function. A user can issue a command via the control connection to the server to indicate from which checkpoint would it like to start the downloading procedure. The server will then begin to transfer the file from that checkpoint when the user gives the start downloading command. A typical procedure is as follows:

1. A user sends command "`REST 10000`" to the server indicating that it wants to download the file from the checkpoint 10000.

2. The server gets the command and prepares to transfer file from the 10000^{th} byte. Then the server responds the user with a "`350 Restarting at 10000. Send STORE or RETRIEVE to initiate transfer.`"

3. The user proceeds by sending a command "`RETR targetfile`" and the server begins sending the targetfile from the checkpoint.

There are many FTP commands. Following table introduce some of them that will be used in this thesis.

USER	The argument field is a Telnet string identifying the user. This command together with the PASS command authenticates a user to access the files on the server.
PASS	The argument field is a Telnet string identifying the password.
PORT	The argument is a HOST PORT specification for the data port to be used in data connection. The fields are separated by commas. A port command would be: PORT h1,h2,h3,h4,p1,p2 where h1 is the high order 8 bits of the internet host address. The p1 and p2 are the local TCP port number, represented in 8-bit decimal format.
TYPE	The argument specifies the representation type. Mostly it is set to "TYPE I", which is used to transfer binary files.
RETR	This command causes the server to transfer a copy of the target file.

REST	The argument field represents the checkpoint at which file transfer is to be restarted. This command does not cause file transfer but skips over the file to the specified data checkpoint. This command shall be immediately followed by RETR command.
SIZE	The argument specifies a file name. This command returns the size of the file. If the file does not exist, a response with code 5yz will be returned.

The response of a FTP server is sent via the control connection too. The response starts with a three-digit code, followed by a human understandable sentence that explains the response. The leading code has predefined meanings. Therefore, a DFT client can judge the response by the code. The meanings of the first digit of the code are as follows:

1yz: The requested action is being initiated; expect another reply.

2yz: The requested action has been successfully completed.

3yz: The command has been accepted, but a further command is required.

4yz: The command was not accepted and the requested action did not take place, but the

error condition is temporary and the action may be requested again.

5yz: The command was not accepted and the requested action did not take place.

2.2 Checkpoint Resume Applications

Many FTP applications take advantage of the FTP checkpoint restart property to realize file-downloading resuming. Some examples are GetRight [Headlight], Download Accelerator [Speedbit], Net Vampire [Afreet] and Gozilla [Radiate]. Most of them are freeware or shareware.

They are actually FTP clients that support checkpoint restart procedure. When the link between the FTP client and server breaks, they will remember the breakpoint. When the link is recovered again, they can automatically resume the file transfer from the breakpoint.

The advantage of these applications is that they can recover the file transfer process from broken link, client crash, or server reboot. Most of the time the recovery procedure will take place automatically. However, there are the following shortcomings:

1. All of them are implemented on the client end only. The users still have to find the server and the file by themselves manually.

2. Checkpoint resume applications do not take advantage of multiple file servers that already exist.

3. If the file server is down, the client does not switch to other available servers automatically.

2.3 Round-Robin DNS

Round-robin DNS [Brisco1995] is a fixed configuration method that can distribute traffic among many hosts. The mechanism of round-robin DNS is as follows:

1. A DNS has pre-configured multiple IP addresses associate with a given domain name.

2. When a domain name resolve request is received by the DNS, it responds with one of the multiple IP address. The selection of IP address is based on a round-robin manner.

For example, if there are three servers that have the same content. The server's addresses are 10.0.0.1, 10.0.0.2, and 10.0.0.3. In the DNS, it associates all these three address with the domain name *www.rrd-example.com*. The first request to resolve *www.rrd-example.com* gets the machine at 10.0.0.1, the second at 10.0.0.2, the third at 10.0.0.3 and a fourth will get 10.0.0.1 again.

Round-robin DNS is easy to configure and implement. It can perform load balancing on Internet traffic including web traffic as well as FTP traffic. However, it has no idea of the load of a server. If a server is over loaded or removed, the DNS will still direct traffic to it according to its round-robin schedule.

2.4 Internet Load Balance Solutions

There are many sophisticated load balance solutions for Internet traffic. Some of them are implemented in software and some in dedicated hardware. Some examples of these solutions are "Content Smart Web Switching" of ArrowPoint [APa] [APb], the "BIG IP" of F5 [F5]. The common properties of all these solutions are:

1. These solutions have awareness of the load, speed, geographic position, and many other performances parameters of the servers.

2. They are intelligent enough to distribute all types of Internet traffic among the servers according to the server's performance.

3. Most of the solutions have the concept of application layer flow switching, which monitors the application layer packet header to make switching decision.

These solutions are quite complex and most of them require special software and

hardware installations. They may also require agent software installed on servers so they can monitor the performance parameters of servers. They do very good job on load distributing. However, most of them are quite expensive and they can only do load distribution within an Intranet.

2.5 Comparison of DFT and Other Solutions

DFT is aimed at providing a reliable file transfer mechanism in the Internet. This goal is realized by connecting a DFT client with multiple file servers. DFT is not a solution on the client end, but a solution on both the server and client ends. DFT has awareness of multiple servers' availability and speed. LDS distributes load among FTP servers. Furthermore, DFT makes use of the existing FTP servers and mirror sites to provide the server "cluster", which makes the cost of DFT very low.

Although DFT does not provide as good of load balance functions as the solutions introduced in section 2.4. DFT is still very good at failure recovery during file transfer. This is the main purpose of our design.

Summary

This section introduced some applications and solutions that increase the reliability and efficiency of file transferring on the Internet. They are all related works that have

similarity with the DFT project. However, DFT has unique properties as presented in the last section. It maintains multiple connections with multiple file servers to increase the reliability of file transfer. That is why it is of interest to design and implement DFT in this thesis.

Chapter 3
Distributed File Transfer Architecture

This chapter presents the architecture of DFT that was developed in this thesis. Section 3.1 describes the outline of the overall system architecture. Section 3.2 introduces the two-layer model of DFT. Section 3.3 describes the working procedure of DFT and section 3.4 gives the fault tolerant mechanism of DFT after introducing a three-layer failure model.

3.1 Distribution Architecture of DFT

DFT has a distributed architecture. It contains three components, *DFT clients*, *a Load-Distributing Server (LDS)*, and multiple *file servers*. Figure 2 shows a typical layout of these three components. The distributed architecture of DFT makes it suitable for reliable file transferring.

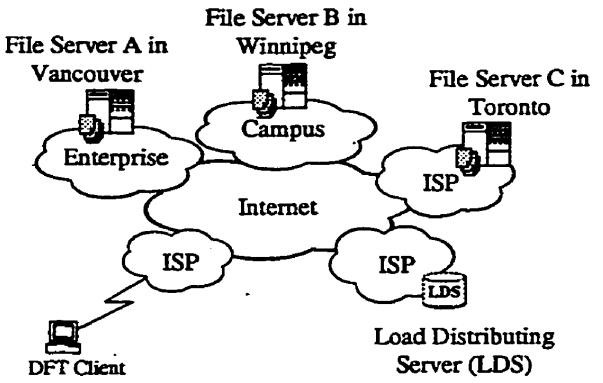

Figure 2 Typical DFT System Layout

DFT's distribution properties can be described through different aspects. DFT has the following distribution properties:

Data Distribution – The data in this thesis refers to the file a user is going to download. In order to achieve reliability, a file is replicated on many file servers. Each file server supports segment transfer of the file. Segment transfer means a user can request an arbitrary length of segment of a file from a file server. Thus, a user can request different segments from different servers and assemble those segments together to get an integrated file.

Geographic Distribution – All components in DFT system can be distributed across the Internet. In Figure 2, there is one DFT client, one LDS and three file servers. We

illustrate the geographic distribution by placing the file servers, which carry the data, in three different cities. In addition, the client and LDS could be anywhere. Geographic distribution is a key property that makes DFT reliable.

Heterogeneous Systems – DFT can run on different platforms. In our thesis, DFT clients and the LDS are implemented in Java, which generates platform independent codes. Therefore, DFT can run on Windows, Linux/Unix, Mac OS or any other operating system that support a Java virtual machine

TCP/IP Connectivity – DFT requires all components connected with each other via TCP/IP. The types of connections to the Internet do not matter. We can assume our client connects to the Internet through a dial up line; the LDS is through an OC-3 ATM connection, the file servers through fast Ethernet. They connect to the Internet via different ISPs. The only requirement for DFT is all parts of the system communicate with each other via TCP/IP.

The following sections will give a brief description on each component of DFT.

3.1.1 DFT Client

A DFT client is a software application that runs on an end user's computer. When a DFT

client downloads a file, it opens multiple TCP connections to a group of servers simultaneously. From each server, it downloads a different segment of the file. The segments are assembled into the target file at the DFT client in a synchronized manner.

A DFT client tests the speed of the link when it connects to a file server. The speed is used to determine the segment size that should be downloaded from that server. If server A is faster than server B and C, the DFT client will request a longer segment from server A.

When a segment cannot be downloaded from the file server, the DFT client will attempt to download that particular segment from another server.

The DFT client sends requests to the load distributing server (LDS) to get information about file servers.

3.1.2 Load-Distributing Server (LDS)

One thing a DFT client need not worry about is where the file is. DFT introduces a new component that dynamically updates the latest file information on each server. It is a Load-Distributing Server (LDS).

An LDS maintains a table that contains information about file servers. The information

includes server names and addresses, directory structures, and the file server's availability. This is important information based on which a client can determine which servers it should choose.

An LDS has two parts inside it. The first one is a file information database, which is learned and maintained while DFT clients request files. The second one is a search engine that can search local and remote databases for a certain file. An LDS provides a uniformed entrance for all clients. From the client point of view, an LDS acts as a file search engine. When a DFT client wants to download a file, it sends a search request to the LDS. The LDS will search for the file in its local database, if it dose not find it, the LDS will search using a public FTP search engine. Then the LDS gives the DFT client a list of file servers, telling the client where the file is. The LDS does not physically contain any files.

The other function of the LDS is to distribute the file transfer traffic according to server and network situations. An LDS collects the information of a server availability through one of two ways: one is information inserted manually by users; the other is inserted by the LDS itself when it finds the target file from the public search engine. Then LDS figures out which group of servers is the "best" for the DFT client to download file from, and tells the client the information about these servers.

3.1.3 File servers

File servers are servers that contain data. DFT clients establish connections with them and download segments of files from them. File servers must have the following properties:

Redundancy – one target file should have multiple copies on different servers.

Segment downloadable – a DFT client can download an arbitrary length of segment of a file from a file server.

In this thesis, traditional FTP servers act as DFT file servers. FTP is a well-defined standard file transfer protocol. FTP servers are ubiquitous in the Internet. The first criterion is met because there are many FTP mirror sites, which contain duplicate files. FTP servers meet the second criterion too. Standard FTP protocol supports checkpoint resume. This means that an FTP client can download from an arbitrary point in the middle of the file, and thus the client can get a segment of the file.

DFT takes full advantage of FTP. DFT turns individual FTP mirror sites into a fully redundant server "clusters" by implementing FTP commands in both our DFT clients and the LDS.

The next section will describe the working procedure of DFT.

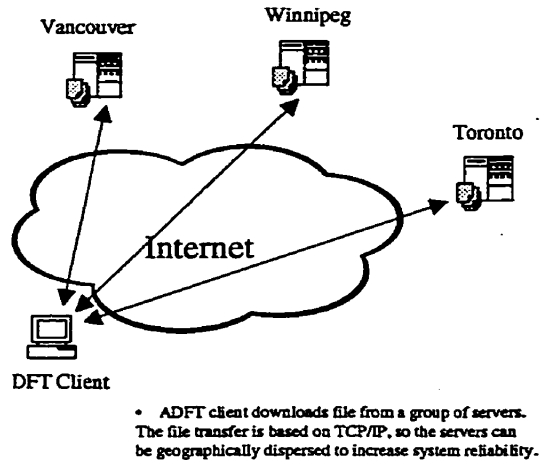

Figure 3 DFT Client and DFT File Servers

3.2 DFT Two-layer Connection Model

Traditional FTP uses two TCP connections. One is a control connection at port 21 to send FTP commands and read responses. The other is a data connection that transfers a file between a client and server.

Figure 4 shows the two connections of FTP session. One connection's function is control, and the other's is data transfer. However, FTP mixes these two types of connections in one server client pair. The FTP server is in charge of both controlling and transferring. If

the server fails, all connections will be lost.

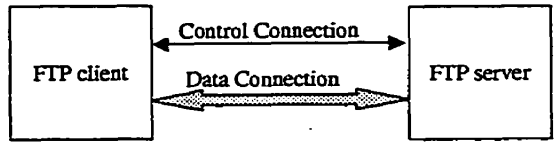

Figure 4 FTP Transportation Model

The FTP model is simple and good for a one-server environment. However, in a distributed reliable file transfer architecture, there are more than one file servers. The control part in the DFT implementation must be enhanced. Therefore, it is a better to separate control connections and data connections. There are two different layers in DFT: the control-layer, and the data layer. The following functions are integrated into the control layer:

1. LDS requests/reply
2. Server status calculation
3. Download process monitoring
4. Download task allocation
5. Failure detection
6. Download resume/recovery

The data layer will concentrate on the following functions:

1. File downloading
2. Status reporting
3. Failure detection

The DFT two-layer model is depicted in Figure 5 and Figure 6. Figure 5 shows the two-layer communication model compared to FTP model while Figure 6 shows different connections belonging to the control layer and data layer.

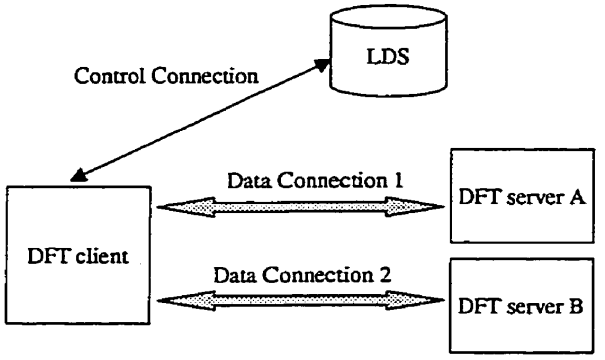

Figure 5 DFT Logical Transportation Model

The control layer and data layer establish their own connections. Connections between a DFT client and an LDS are control layer connections. Connections between a DFT client and file servers are data-layer connections.

The DFT two-layer model separates the data transfer and transfer control parts. With this model, it is possible to establish and maintain multiple data connections between one DFT client and several DFT file servers. With this model, we can enhance both control and data transfer layers separately. Thus, we can provide a more robust file transfer system.

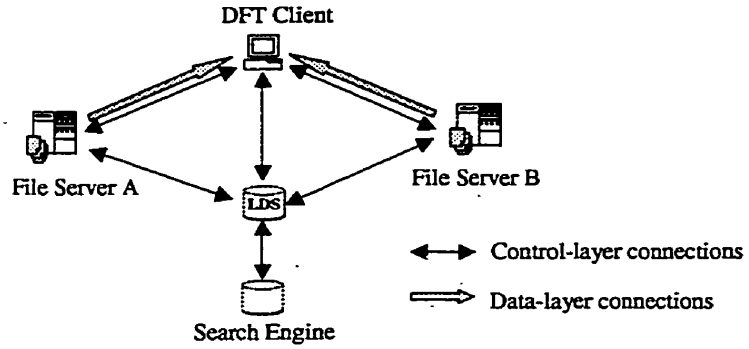

Figure 6 DFT Control-level Connection and Data-level Connection

The following sections discuss these two layers in detail.

3.2.1 Control-layer Connection

The purpose of a control-layer connection is to control and maintain data connections. Before a DFT client begins downloading a file, it does not know where the file is.

Though a control-layer connection, the client makes an enquiry at the LDS and learns where the target files are located and the condition of the file servers. Then the client can start data level connections with those servers and start the data transfer.

When data transfer is in progress, the client will report information to the LDS about the link to the server. This information is used to help the LDS calculate the server's performance and the link's speed. This status reports will also be transferred to the LDS through control-layer connections.

Generally, control-layer connections are established between a DFT client and an LDS. However, when a file cannot be found in an LDS's local database, the LDS will connect to a file search engine to find that particular file. This kind of connection is concerned with file information retrieving. We classify these as control-layer connections.

Periodically, an LDS will poll file servers in its database to know if they are available. This is done by establishing connections between the LDS and the file servers. This will help the LDS provide more accurate information to DFT clients. This type of connection is a third type of control-layer connection.

Control-layer connections are responsible for starting, stopping and resuming data connections. The download control thread in the control-layer keeps monitoring the

situation of all data connections. In the case when data connections stall or fail, control-layer connections will reestablish other data connections to substitute the compromised ones. This isolation and enhancement of control connections ensures the reliable transfer within DFT.

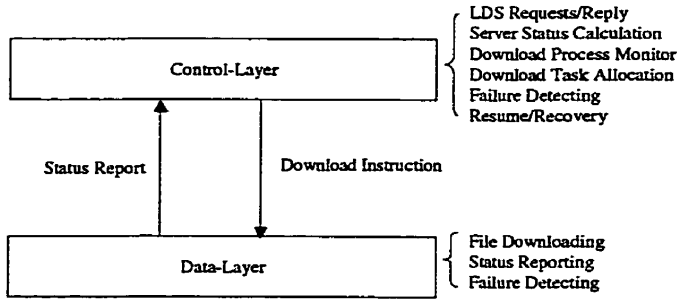

Figure 7 DFT Two-layer Model and Their Functions

3.2.2 Data-layer Connection

Data-layer connections are established when the "real" part of DFT comes to work. When a client downloads a file from DFT file servers, it establishes data-layer connections between itself and the file servers. The target file is transferred through data-layer connections. One data-layer connection is responsible for one piece of a file segment. It is not concerned about the file's availability, nor is it concerned about failure and recovery.

All a data-layer connection needs to do, is download the segment of the file under the instructions of a control-layer connection.

However, while data connections are transferring data, the control layer keeps monitoring each data connection. As soon as a control layer detects a data connection's unusual behavior, it will adopt an appropriate action. A data connection is responsible for reporting any failure to the control layer.

3.3 DFT Working Procedure

Just as the DFT model has two layers, the working procedure of DFT has two steps:

1. Target file query and searching
2. Multi-connection file downloading

3.3.1 Target File Query and Searching

Target file query and searching is the main task of the control layer. When a DFT client wants to download a file, it will establish a control layer connection to the LDS. Through this connection, it sends a request to the LDS. The request includes a query, which asks the LDS information about the target file. The LDS receives the request and searches its local database. If it finds the related record about the file, it will respond to the client with

the corresponding information. The information includes the file name, file servers' addresses, and file locations. After the client get this response from LDS, it can start file downloading.

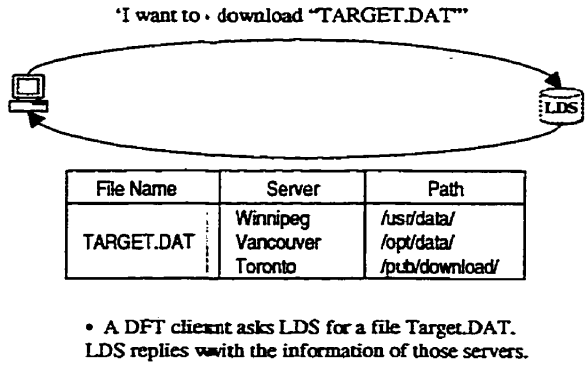

Figure 8 Queries and Reply between a DFT Client and an LDS

If the LDS cannot find an entry for the requested file in its local database, it will send a query to an FTP search engine. There are many FTP search engines publicly available. One well-know search engine is Lycoos FTP search engine [Lycos]. In our implementation, we use this one. After receiving the reply from the FTP search engine, LDS will add the reply to its own database and at the same time, respond to the DFT client with the new added record.

If nothing is found in both the local database and the public search engine, a negative reply will be sent to the DFT client. The file downloading procedure will be terminated.

3.3.2 Multi-connection File Downloading

File downloading is the main function of the data layer. The data layer gets instructions from the control layer and performs data transferring between a DFT client and file servers.

Once the DFT client gets a positive answer from the LDS, it sends downloading instructions to the data layer. Downloading instructions come from the LDS reply. The DFT client parses the reply and knows which servers contain the target file. For example, in Figure 8, three servers contain the file. Then, three downloading instructions will be passed to the data layer. The DFT client's data layer then tries to establish three data connections with all three servers. Each data connection is actually an FTP connection.

After data connections are established, the client will test the speed of the connections as well as the availability of the target file. The speed is simply tested by sending FTP "`SIZE`" commands and calculating the elapse time between sending the command and receiving the reply. The response of "`SIZE`" is the file length in bytes. If the file is not on the file server, the response will begin with an error code. Thus, the FTP "`SIZE`"

command and reply tests both the server's access and the file's availability.

According to the speeds of the servers, DFT client calculate the length of segment from each file server it should download. DFT uses the following formula to calculate the segment length l_i:

$$l_i = \frac{s_i}{\sum s_i}, \quad i = 1, 2, 3, \ldots, N,$$

Where s_i is the speed of server i, and N is the number of servers.

The segment length is calculated in such a way that all data connections should last approximately the same amount of time. This will maximize the overall downloading performance. However, the server's speed is a variable that changes from time to time. As such, the speed variable is just an estimate. Even with this approximation, performance of downloading is greatly improved and likely near optimal. Chapter 5 will present the performance tests of DFT.

The DFT client starts to download each segment from each file server simultaneously once the segment lengths of all servers are determined. The client synchronously writes each segment into the local file.

While downloading from each server, all data connections have a timeout value set. When there is no data from the server within the timeout period, this data connection will terminate its downloading and report to the control layer. The control layer will tell the data connection of other available servers (if there are any) for this data connection and this data connection can resume its downloading task.

Figure 9 shows a DFT client downloads three segments from three file servers.

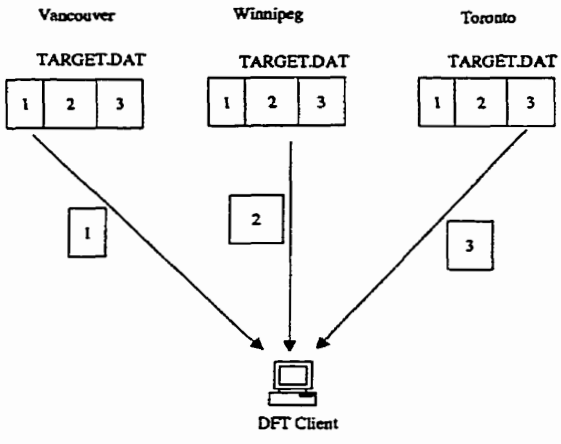

Figure 9 Multiple File Segments Downloading

3.4 Fault Tolerance

Before presenting the fault tolerance feature of DFT, this section will discuss a

classification of faults and corresponding reasons for their occurrence. A layered failure model is introduced in the first section. It will help in detecting a failure and the recovery from it. After that, a failure recovery technique will be described.

3.4.1 Layered Failure Model

A failure model is very helpful to detect and recover from failures. For reference, a communication network has a seven-layer OSI (Open Systems Interconnection) reference model. Similarly, to analyze failures, we define a three-layer failure model. See Figure 10 for the comparison of the two models.

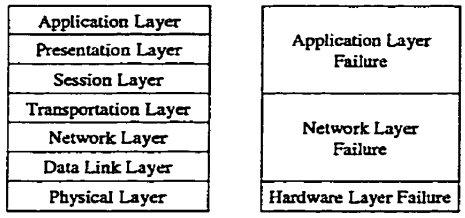

Figure 10 OSI Reference Model and Three-layer Failure Model

The three-layer failure model has the following layers:

Application layer failure – Application failures are the failures that can be positively detected at the application level of DFT client. These kinds of failures are normally

caused by the application itself. The reasons for application failures are normally file server malfunctions. For example, when an FTP server has too many users logged in, it will ban new users and respond with a "421" error code. This will terminate a data connection and thus the corresponding segment cannot be downloaded. A failure such as a login failure, authentication failure, and destination file's availability failure all can be detected as soon as the file server responds with a negative reply. These are typical application layer failures.

Application failures do not tear down network connections. When application failures occur, the communication between DFT client and file server is still active. The DFT client can still send commands to and read responses from the file server.

Network layer failure – Network failures are the failures associated with a malfunctioning network or those failures that cannot be positively detected by the DFT client. Such failures do not send any message to the client. A DFT client can detect a network failure when its TCP connection is broken or a preset timeout is fired. Most network failures happen because of network problems, such as network congestion, link failure, etc. However, sometimes, a DFT client cannot know the reason of a network failure. To a DFT client, all timeout events look the same. When a timeout occurs, the DFT cannot tell whether it is because the network is too busy or the server at the other end has stalled.

Hardware layer failure – Hardware layer failures are the failures that occur on the DFT client side. They are caused by a DFT client's malfunction. The reasons include a client's power failure, a full hard drive, a network interface being down, an operating system crash, etc. This type of failure generally causes all downloading connections to be terminated. If the failure is caused by a power failure, the entire client will be killed. When this kind of failures occurs, the DFT has to recover its job from the break point recorded by the client before it stopped.

The three-layer failure model is defined from the DFT client point of view. Thus, an abnormal crash of a remote server has the same effect as a physical link break. Although to the server, it is obviously a hardware failure, but to the client, the only symptom is TCP connection timeout. This is the same as a network failure. However, if the file server gives alarms before it halts, this failure will be detected by the DFT client and the failure is classified as an application failure.

Next section will discuss how to detect the failures and the appropriate recovery action in DFT implementation.

3.4.2 Failure Detection

Failure detection can take place on both the control layer and data layer. When the control

layer detects failure, it will try to recover from it immediately. When a data layer detects a failure, it will stop its current job and notify the control layer.

When failures occurs, we have to detect them as soon as possible. In the DFT system, we have two ways to detect a failure:

1. Failure code detection
2. Timeout detection

Failure code detection is used to detect application layer failures. When application layer failures happen, a DFT client still has connection with the file server. It detects the failure by parsing the response from the other end of the connection.

When a DFT client makes a connection or sends a command to a file server, it will receive a string as a response. The string will begin with a three-digit number followed by a sentence explaining the response. A typical response is like this: "`226 Transfer complete.`" As we discussed in Chapter 2, we can judge the response by observing the three-digit code. A code larger than 400 generally indicates an incomplete service. If a client reads such code from the server, it knows there is problem. In Appendix A, we give a file server response code and explanation list.

Through failure code detection, all application failures can be detected immediately. Such

failures include a) server not available, b) file not found on server, c) too many users, d) login failures and, e) failure to establish a data connection.

Timeout detection is a common method when detecting network failures. When network failures occur, there will be no messages transferred through the connection. TCP timeout makes the connection retransfer. However, this does not work if somewhere inside the network is down. To avoid this problem, a DFT client sets timeout on each connection. When the timeout fires, The DFT client assumes that the connection does not work and switch the task associated with it to another connection.

There are two types of timeouts, the command-response timeout and the data timeout. Command-response timeout is used on control connections. After a DFT client sends out a command, it starts the timer. If timeout fires before it receives the response, the client stops this control connection. Data timeout is set on a data connection. We set a timeout in each data connection socket. If the elapse time between two packets is longer than the timeout, the client assumes the data connection is down, or the network is becoming congested.

Through timeout failure detection, network failures can be detected.

In spite of failure code and timeout detection methods, other common failure detection

methods are also used in our implementation. For example, if a TCP pipe is broken before our preset timeout fires, the DFT client also detects it and treats it as a timeout event.

3.4.3 Failure Recovery

Failure recovery is the work of the control layer. When the control layer detects a failure or receives the failure report from the data layer, it will do the following steps:

1. Record the latest status of the failed connection.
2. Terminate the failed connection.
3. Choose another connection that can do the terminated job.
4. Switch the job to that connection.

To improve performance, in step 3, the DFT client will choose the fastest server to switch the terminated job. The speed of server is tested at the beginning of a connection.

This four-step recovery procedure is suitable for all application and network failures. However, when hardware failures occur, the DFT client itself is terminated. Since the computer on which the DFT client is running crashes, human intervention is required. The downloading job can only be resumed when the DFT client runs again. To recover from such a failure, we have to record the status of all connections on a hard drive. Given

the hard drive is readable when the DFT client comes to life again, the client recovers all connections by reading the status record. In the implementation, a DFT client periodically (every second) records the status of all connections' on the hard drive. Through this method, hardware failure at the DFT client can be recovered as well.

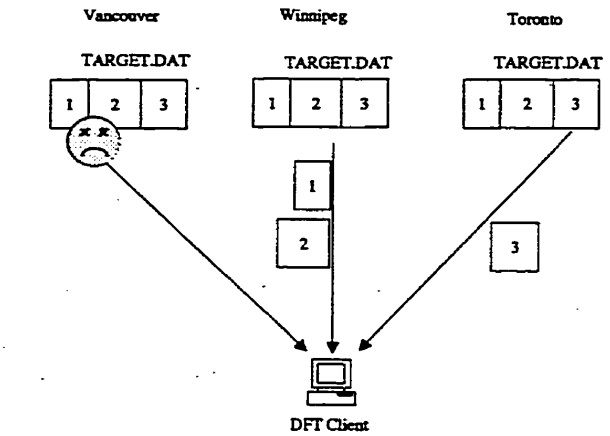

Figure 11 Job Switching when Failure Occurs in DFT

Summary

This chapter has presented the architecture of the DFT that we developed for this thesis. We introduced the distribution structure and the three parts of the DFT system. Following that, we presented the two-layer working model of the DFT. We detailed the working procedure of DFT. As for failure recovery, we introduced the three-layer failure model

and described the recovery procedure of our system. The next step is to detail the design of the DFT client, LDS and file server.

Chapter 4
DFT Implementation

This chapter gives a detailed implementation of a DFT client and an LDS. Section 4.1 introduces the implementation tools. Section 4.2 describes the implementation of the DFT client and Section 4.3 describes the implementation of LDS.

4.1 Implementation Tools

DFT is designed to be platform independent. The implementation tool must meet this goal. We chose Java as our programming language to implement the DFT client and the LDS. The GUI interface of the DFT client is implemented in JFC. As for the DFT server, we use existing FTP servers.

4.1.1 Java

Java was introduced by Sun Microsystems in 1995. One of the design goals of Java is to

provide a programming language that can develop applications suitable for distributed computation environments. All java applications can run on different platforms. This portability makes Java widely accepted as a programming language in developing Internet applications. Since its introduction, Java has been widely supported. All major Web browsers include a Java virtual machine and almost all major operating system developers (IBM, Microsoft, and others) have developed Java compilers to support Java.

The major features of Java are:

- Java is object oriented, and very simple. In order to take advantage of modern software development methodologies and to fit into distributed client-server applications, Java is designed as an object oriented language. Java has similar syntax as compared with C/C++. Therefore, developers familiar with C/C++ can quickly adapt to Java and find it is easy to learn and use.

- Java is an interpreted language. This feature makes the development cycle much faster. Developers need just compile and run their code. Link and load steps are not issues in Java. Furthermore, Java's just-in-time compiler can dynamically compile Java bytecode into executable code. This tremendously improves Java application's performance.

- Java applications are portable across multiple platforms. Java applications never need to be ported — they will run without modification on multiple operating systems and hardware architectures. A Java program is complied into bytecode. The interpreter, Java virtual machine (JVM), interprets Java bytecode and executes it on a computer's hardware at run time. JVM hides all differences between different hardware and operation systems from the Java application. This is why Java can create applications that are denoted "Write Once, Run Anywhere".

- Java applications are robust. The memory management of Java is not a burden for the developer. The Java run-time system takes care of all memory management tasks. This feature helps Java applications avoid ill-formed memory operations. Unlike programs written in C/C++, Java has no pointers. This means an instruction in Java program cannot contain the address of data storage in another application or in the operating system itself. Both situations can cause poor memory allocation and release operations, which in turn cause system to terminate or "crash".

- Java has built in support of multithreading. This feature improves the performance in applications that need to perform multiple concurrent activities, such as multimedia and GUI.

- Java applications are adaptable to changing environments because developers can dynamically download code modules from anywhere on the network.

- Java applications are secure. This feature is necessary for an application that needs to run across the Internet. The Java run-time system has built-in protection against viruses and tampering.

In summary, the Java Programming Language platform provides a portable, interpreted, high-performance, simple, object-oriented programming language and supporting run-time environment. For these reasons, it was selected for this project.

4.1.2 Java Foundation Classes (JFC)

The Java Foundation Classes (JFC) is a significant cross-platform graphical user interface component and services solution. It was first introduced with the Java Development Kit (JDK) software release 1.1 and was a joint work of Sun, Netscape, and IBM. Now JFC is delivered as part of the Java platform. It can be used to develop large scale, mission-critical intranet and Internet applications.

Java Foundation Classes is based on the original AWT. It also includes many of the key features of Netscape's Internet Foundation Class (IFC). It extends AWT by adding a

comprehensive set of graphical user interface class libraries and is compatible with all AWT-based applications.

JFC has the following features:

- JFC is core to the Java platform and reduces the need for bundled classes.
- All new JFC components are JavaBeans architecture-based. JavaBeans is an architecture and platform neutral API for creating and using dynamic Java components. Therefore, developers can create more compatible and portable Java applications and applets with JFC
- No framework lock-in. Developers can easily bring in third party's components to enhance JFC applications.
- JFC components are cross-platform.
- JFC subclasses are fully customizable and fully extendible.

4.2 DFT client's implementation

4.2.1 DFT Client's Function Block

A DFT client runs on a user's computer. It responds to a user's input and carries out

appropriate actions. Briefly, it has three function blocks: *download threads*, a *download thread controller* and a *logging thread*. Figure 12 shows the data flow and control flow between these function blocks.

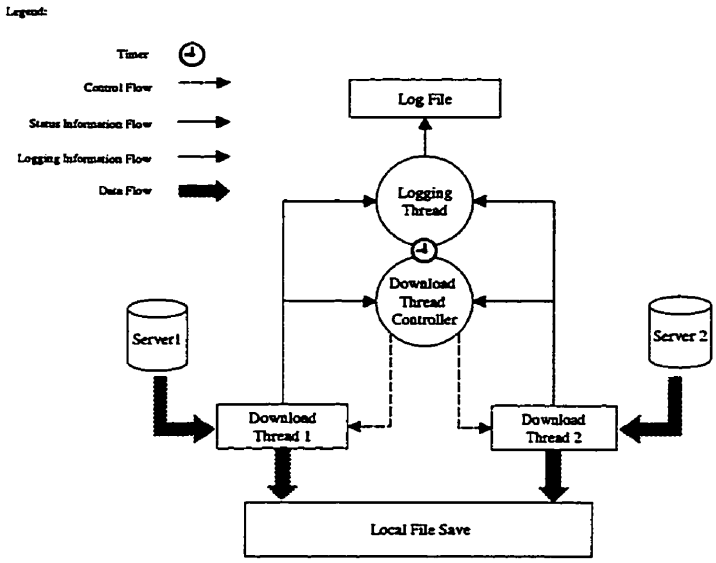

Figure 12 Function Blocks of a DFT Client and the Information Flow between Them

Download thread – Each download thread takes care of a file transfer session with one file server. A DFT client may have one or several download threads. Each thread downloads a segment of the destination file from a file server and writes it into the local save-as file. When all download threads finish successfully, the file downloading is

finished successfully.

The functions of a file download thread include file server log in, file segment request, and downloading status report for the logging thread and controller thread. A download thread stops if one of the following conditions is met:

1. The number of bytes it downloaded equals or exceeds the segment length. This means the download thread finishes successfully.

2. There are failures stops the download thread.

Once a thread stops downloading, the corresponding file download thread sets its finish-flag. The download thread checks how many bytes it has downloaded. If the length is less than expected, the segment is not complete. The download thread sets the stall-flag. The download controller thread can check these flags. If the finish-flag is set and the stall-flag is not set, the segment is completed successfully. Otherwise, if the finish-flag is set and stall-flag is set too, the segment is stopped but not completed.

Table 1 shows the combinations of finish-flag and stall-flag and their meanings.

Finish-flag	Stall-flag	Meaning
0	1	Not defined
0	0	Downloading is in progress
1	0	Downloading finishes successfully

1	1	Download finishes but this segment is not complete.

Table 1 Meanings of Finish-flag and Stall-flag combination

Download thread controller – The download thread controller makes sure that a DFT client can resume its work when some file servers are not available. A download thread controller periodically monitors the status of all download threads. When it finds a download thread is stalled (finish flag set but number of downloaded bytes is less than segment length), it will select another active file server and switch the stalled download thread to that server.

Logging thread – The logging thread periodically polls all download threads for their latest progress. It queries the information on the file server's address, segment length, beginning position of the segment, and how many bytes the download thread has already downloaded. It also queries the finish and stall flags of the download threads. Then, it writes all the information into a log file on local hard disk.

If all download threads are terminated, the DFT client can check the log file to know where should it resume the work. Therefore, even when the DFT client's computer crashes, the download work can still be resumed from where it stopped.

The following flow charts and explanations show the detailed working procedure of a

DFT client. They are organized in following order:

Figure 13 shows the overall download procedure of a DFT client. Table 2 explains each block. Figure 14 shows the working procedure of a controller. Table 3 explains each block. Figure 15 shows the working procedure of a download thread. Table 4 explains each block. Figure 16 shows the working procedure of the timer. The timer is used by both the controller and the logging thread. Table 5 explains each block.

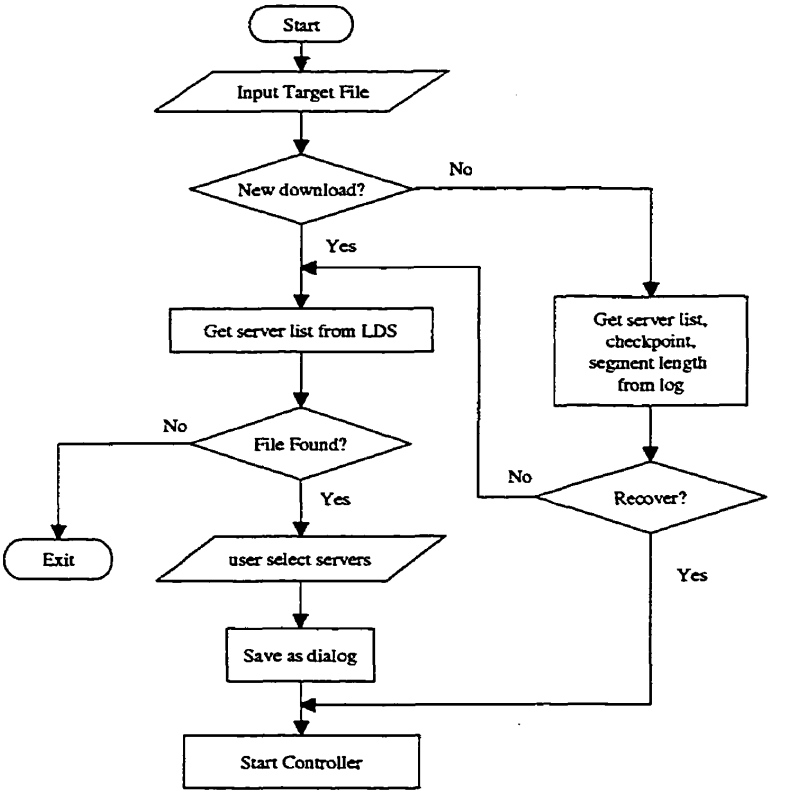

Figure 13 Overall Download Procedure of a DFT Client

Input target file	This dialog asks the user which file he wants to download.
New download?	Check if the log file exists; return false if it exists, true if not. The log file is created and updated when a download is in progress. If the download process finishes successfully, the log file will be deleted. Otherwise, it remains on the hard drive.
Get server list from LDS	The DFT client sends a search request to the LDS, the LDS searches its database, if it finds the location of the target file, it will reply to the client with a list of servers that contains the file. Otherwise, it will reply to the client with "Not Found".
File found?	Reads the LDS response and checks whether the response contains a server list or not.
Get server list, checkpoint, segment length from log file	If the log file exists, the DFT client reads and parses the log file. Then it knows where the target file is located and the breakpoints where the last download process terminated.
Recover dialog	This dialog lets the user confirm the recover from a terminated downloading process. If the user chooses not to recover, the download process will start from beginning. Otherwise, the download thread will resume from where it left.
User select servers	This dialog gives user a list of choices on which server(s) he would like to use. The DFT client will download from the servers the user has chosen.
Save as dialog	This dialog lets users choose where he want to save the target file.

Start Controller	Start a new download thread controller.

Table 2 Explanations of DFT client's overall behavior flow chart

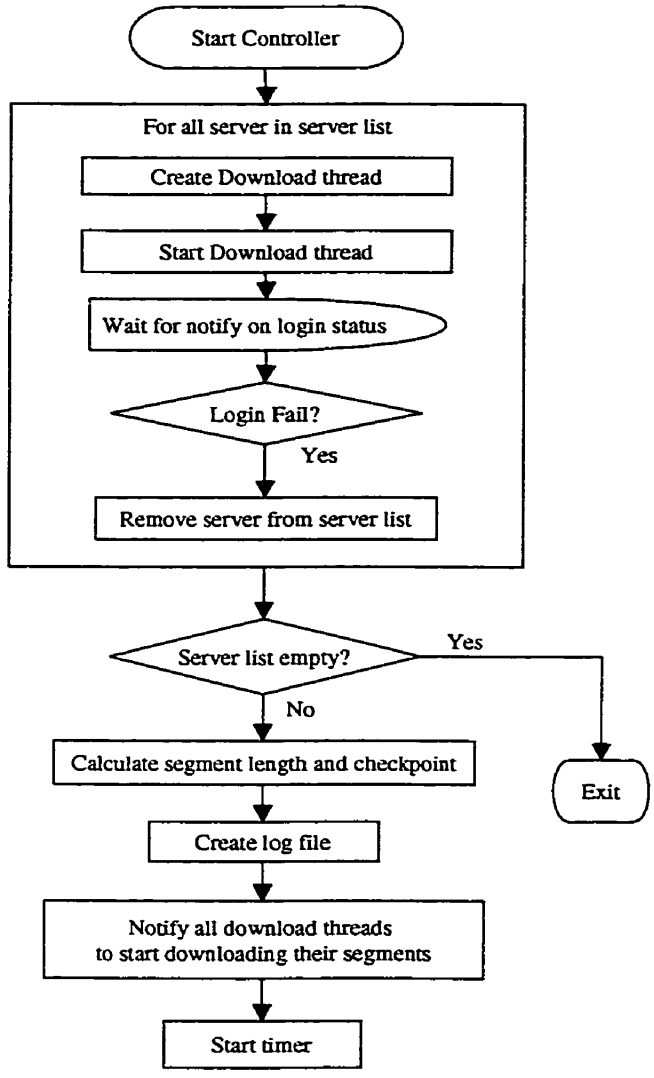

Figure 14 Working Procedure of a Controller

Create download thread	For each server in the server list, create a download thread with the server's address, user name and password.
Start download thread	Let the download thread try to log into the server with the username and password.
Wait for notify on login status	If the download thread cannot login to the server, the download thread will notify the controller with a login failure. This situation maybe due to the server not being reachable, too many users on the server, or authentication failure. Otherwise, a download thread notifies the controller with a login success.
Remove server from server list	If login fails, the server is no longer available for this downloading procedure. Therefore, it is removed from the server list. Furthermore, as the download thread associated with this server is useless, it is deleted too.
Server list empty?	If no servers are available, the server list will be empty. The downloading procedure cannot carry on any more and the download procedure is terminated.
Calculate segment length and checkpoint	At this stage, all download threads associated with the servers in the server list have successfully logged in and obtained the server's speed. Therefore, the segment length of each download thread can be calculated according to different server speeds. The faster the server is, the larger the segment is. After all segment lengths are calculated, the checkpoint of each segment is easily obtained. Later, all segments will be inserted in the local save-as file at the corresponding checkpoint.

Create log file	A log file is created in the working directory. Its file name is "target file.log".
Notify all download threads to start downloading their segments	After the segment length and the checkpoint are calculated, the download thread is ready to download the segment. The controller notifies the download threads to start downloading. From this stage on, a real data transfer procedure commences.
Start timer	The controller starts a timer. This timer will check the status of all download threads periodically. In addition, it will log the progress of each download thread.

Table 3 Explanations of DFT client's download thread controller flow chart

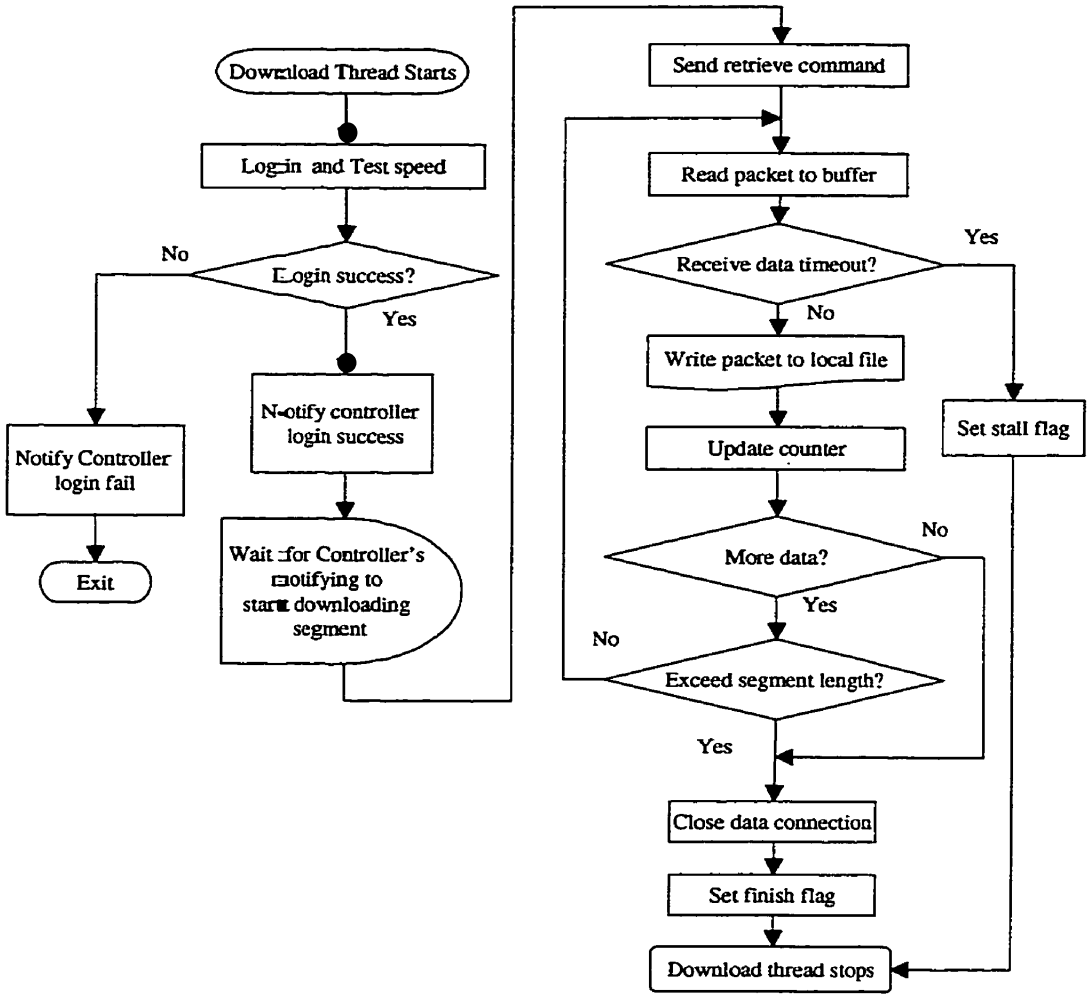

Figure 15 Working Procedure of a Download Thread

Log in and test speed	The download thread logs in the FTP server. If the login is successful, it sends FTP SIZE command to the server. Then the download thread records the response time. Speed is calculated by dividing the response length by the response time.

Login success?	True if login success. False if not.
Notify controller login success	If there is a login success, the download thread's login method returns a Boolean true. The controller can read this return value.
Notify controller login fail	If login fails, the download thread's login method returns a Boolean false. The controller can read this return value. The download thread will terminate itself.
Wait for controller's notification to start downloading a segment	Waits for the controller to calculate the segment length and checkpoint. Download threads cannot start downloading if the checkpoint and segment length are unknown.
Send retrieve command	Sends retrieve commands to the file server and starts to receive data. This step includes opening a data socket to receive FTP data.
Read packet to buffer	Reads arriving data packet from data socket.
Receive data timeout?	If a socket timeout event occurs, the connection to the server is assumed lost. The server either becomes too slow or is no longer available. This is a failure. The download thread sets its stall flag and closes all data connections. It now is waiting for the controller's instruction.
Write packet to local file	The download thread locks the local file from other download threads' writing. It moves the file's write pointer to the last position it left and continues writing the packet into the local file. After writing, the download thread unlocks the local file.

Update counters	The counters include the overall bytes all download threads have downloaded, how many bytes this download thread downloaded and the position where the write action left.
More data?	If the socket's input stream ends, there is no more data.
Exceed segment length?	If the number of bytes this download thread downloaded is larger or equal to the segment length assigned to it, it has already finished its segment.
Close data connections	Close the FTP control connection and FTP data connection to the server.
Download thread stops	The download thread stops. If it is stalled (stall flag set), it is waiting for the controller's further instructions.

Table 4 Explanations of DFT client's download thread flow chart

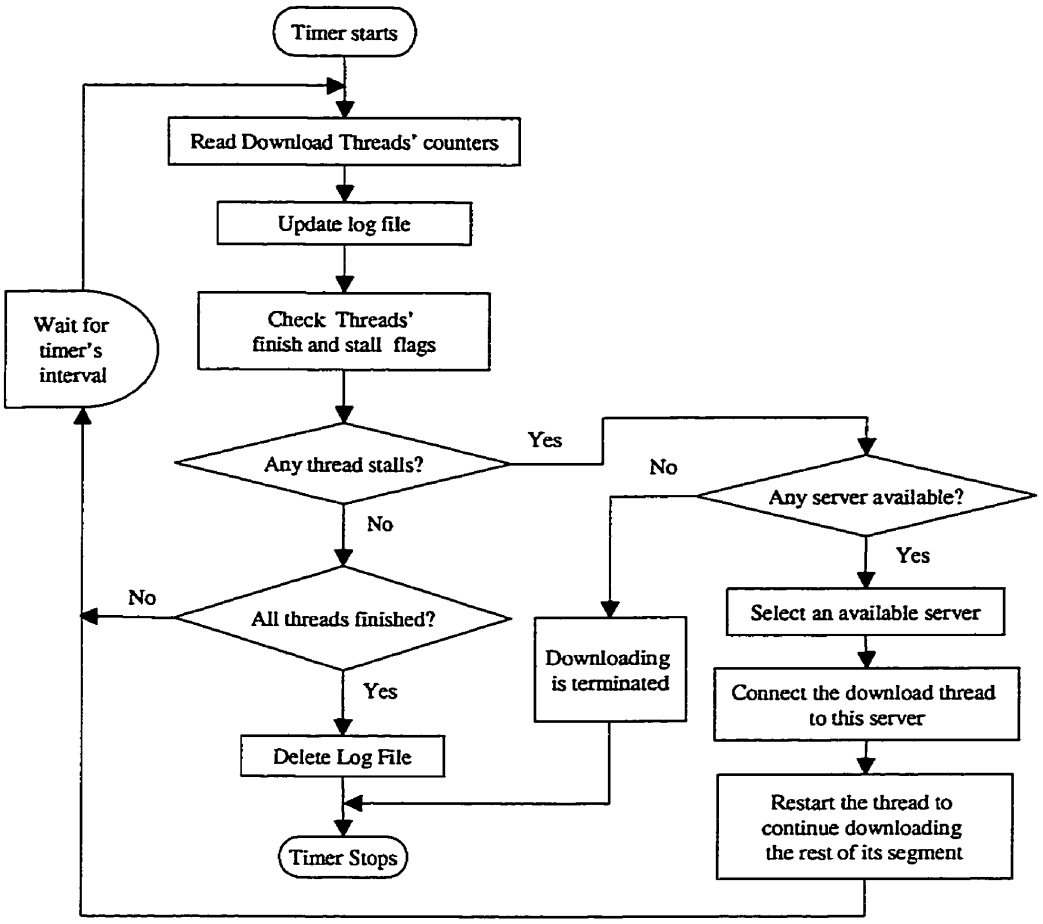

Figure 16 Working Procedure of the Timer

Timer starts	Timer's default interval is set to 1 second
Read download threads' information	Read information from every download thread. The information includes: server address, username, password, target file name, target file path, target file length, checkpoint, segment length, and the number of bytes that have been downloaded.

Update log file	Update the log file with the information just read.
Check download threads' finish and stall flags	Check the download threads' finish and stall flags.
Any thread stalls?	If a stall flag is set, that download thread is stalled.
All threads finished?	If finish flag is set, that downloading thread has finished.
Delete log file	When the downloading successfully finishes, the log file is no longer needed. Therefore, it is deleted.
Any server available?	If a download thread's stall flag is not set, the server associated with that download thread is available.
Select an available server	Get the server's address, username, password, and target file path.
Connect the download thread to this server	Set the download thread associated with this server, i.e., change the server address, username, password, and target file path.
Restart the thread to continue downloading the rest of its segment	Clear the download thread's stall flag and finish flag. Advance the checkpoint by the number of bytes this download thread has already retrieved, less the segment length by the same amount. Set bytes retrieved to zero, then, let the download thread start to retrieve data from the new server again.

Stop timer	The controller and logging process as are stopped.

Table 5 Explanations of DFT client's timer flow chart

4.2.2 Major Classes in DFT Client

ControlConnection

ControlConnection sends commands to the file server and receives corresponding responses. It connects to a file server on TCP port 21, which is FTP control port. If the response shows the server is not available, ControlConnection throws a CommandException.

DataConnection

DataConnection receives data from a file server. It maintains a TCP connection and reads data from that connection. Meanwhile, it writes the data into a local save-as file in a synchronized manner. It also updates the counters with the number of bytes it has written, and the number of bytes all instances of DataConnection have written.

DownloadThread

The DownloadThread uses ControlConnection and DataConnection to download a segment from a file server. It catches exceptions thrown by both of the two classes. It sets

flags to indicate the status of downloading. The flags are stall flag and finish flag. If stall flag is set, the segment has not been downloaded successfully. If the stall flag is not set and finish flag is set, the segment is downloaded successfully.

Getit

The class Getit is a controller class that controls an array of DownloadThreads to download a file from different file servers. It is in charge of getting the server list from LDS, assigning a segment to each server and creating an instance of DownloadThread associated with that server, and making sure all instances finish successfully. The controller provides a method to check the status of all DownloadThreads. It can also write the status into a log file on a local hard disk. If the controller detects that a DownloadThread has stalled, it can switch that thread to another file server to resume downloading the segment.

LDSConnection

The LDSConnection gets the server list from which a target file will be downloaded. It has two methods to get the server list. The first is parsing a local log file. The other is getting a response from an LDS. A log file exists when a downloading procedure was interrupted. If the target file's name is "TARGET", the log file is named as "TARGET.log". It is a simple text file. The file's format is as follows:

```
Line 1  ftp.download.com /pub/win95/internet anonymous a@a.com d:\i.exe
        3007751 0 2197504 0 6450606 0 0
Line 2  ftp.icq.com /pub/icq/win95/internet anonymous a@a.com d:\i.exe
        3204166 2197504 4253102 0 6450606 0 0
```

Each line in the log file stands for an instance of the DownloadThread. The items in a line are separated by a space. The first item is the file server's address. The second one is the path information. The third and fourth are username and password. The fifth is the local save-as file. The sixth is the speed of the server. The seventh and the eighth are the checkpoint and length of this segment. The ninth is how many bytes have already been downloaded by this thread. The tenth is the total length of the target file. The eleventh and the twelfth items are the finish and stall flags of this thread.

4.3 LDS implementation

4.3.1 LDS Function Blocks

LDS waits for connections from DFT clients on TCP port 6666. The communication flow between a DFT client and LDS is depicted in Figure 17.

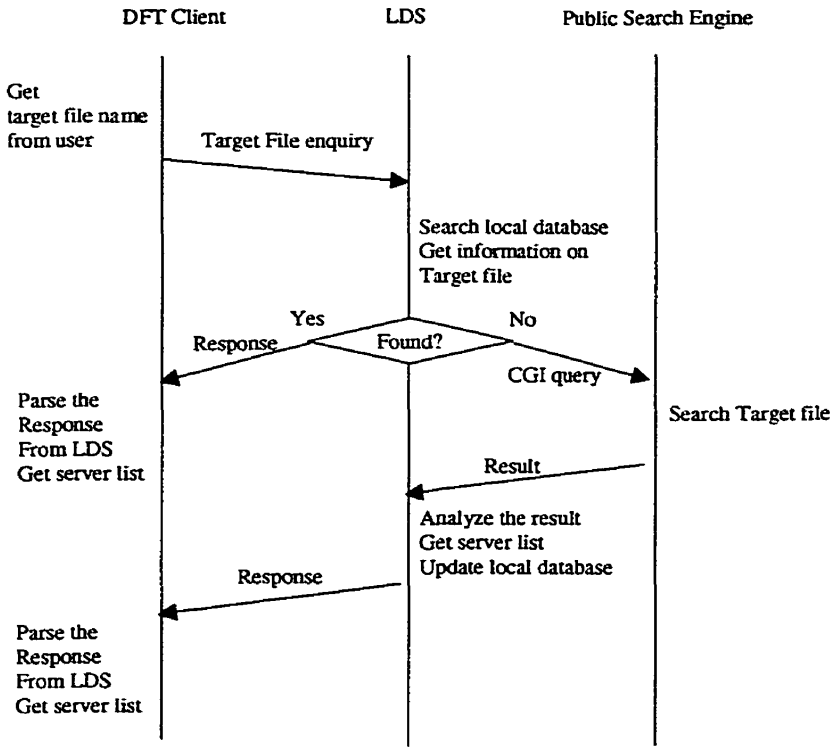

Figure 17 Communication between a DFT Client and an LDS

As shown in Figure 17, there are four steps for a DFT client and LDS to finish communication:

Step 1. A DFT client sends a target file search request to LDS when the client gets a target file name from user.

Step 2. LDS searches its database to get records of the target file.

Step 3. LDS sends the results back to DFT client. The result will be records of file servers that contain the file if the target file is found. If the file is not found in the local database, the LDS will send a query statement to the public FTP search engine. If the file is found in the search engine, the LDS will add the records into its database and send them to the client. Otherwise, the LDS will respond with a string of "Not Found".

Step 4. The DFT client parses the response and begins to download. If the response is "Not Found", the client stops.

The local database in the LDS is a simple text file. Each line in the text file represents a record. A text file was selected as the data source mainly because of its simplicity. Any platform, Windows or Unix, supports text files very well. Each line of the text file has following fields: server address, path that the target file locates, username, password, and target file length. These fields are separated by a space.

The response of an LDS is several lines of text. Each line is a record of a server that has the target file. The line has the same format as the line in the database text file as we described before.

4.3.2 Major Classes in LDS

LDSServer

The LDSServer listens on port 6666 and accepts new connections from DFT clients. When a new connection is accepted, it creates a thread of LDSServerThread to handle the connection.

LDSServerThread

The LDSServerThread reads the target file enquiry and sends the results back after it searches its local database. If it cannot find the target file, it sends "Not Found".

Get

Get tries to send a search query to the CGI gateway of the FTP search engine and analyzes the results. The search engine in our code is the Lycos FTP search engine. The Lycos search engine accepts HTTP GET commands to send CGI commands to its search engine. The GET command is as follows:

"GET

```
/cgi-bin/search?form=lycosnet&query=TargetFileName&doit=Go+G
et+It%21&filetype=All+files"
```

The search engine will reply with an HTTP formatted page that contains the results. Get searches through the results and finds the right information for the file servers.

Summary

This chapter described the detail implementations of the DFT client and the LDS. The DFT clients and LDS are both implemented in Java. The GUI of DFT client is implemented using JFC.

Chapter 5
Experiments and Data Analysis

This chapter gives a series of experiments that explore the features of the DFT architecture. First, the design of the experiments are described, including the goals that the experiments are to achieve and the environments of the experiments. After that, experimental data is collected and analyzed. The data shows both the reliability and performance features of DFT. The last section is the summary of this chapter.

5.1 Experiment Design

The last chapter illustrated the implementation of all components of a DFT system. An experiment is carried out in this chapter to test the features of DFT. DFT has many features, as we described in Chapter 1. However, among them, only the most important features will be tested here. These features are:

1. The reliability of DFT.

2. The performance of DFT.

One of the motivations of designing DFT is to increase the reliability of file transferring. Therefore, the first thing to test is if the implementation meets this goal and how well it fits our requirements. If DFT's performance is poor, it is not practical. Therefore, the experiment must also test the performance of the implementation.

5.1.1 Experiment Goals

Reliability Test – DFT can detect and recover from many kinds of failures. According to the layered failure model developed in Chapter 3, there are application layer failures, network layer failures and hardware layer failures. Different failures lead to different detecting and recovery procedures. The experiment should realize or simulate these different failures. DFT should detect all those failure and recover from them.

The test measures the *overall failure/recover time* when applicable. The overall failure/recovery time is the time elapsed between the failure occurrence and successful recovery. It has two parts. The first part is failure-detecting time, which starts from the time failure occurs and ends at the time the DFT detects the failure. The second part is the

recovery time, which starts from the time when the failure was detected and ends at when it is recovered. The overall failure/recovery time should be very short compare to the downloading time.

The failure/recovery time is applicable when the failure can be automatically recovered. However, if the DFT client is halted or the network interface of the client fails, human intervention has to be performed. A DFT client can only resume downloading after the failure of the system or the network is recovered from. Therefore, the failure/recovery time does not have practical meaning. In these scenarios, the experiment only tests if the failure can be recovered from.

Performance test – DFT is designed not only to increase the reliability of file transferring over the Internet, but also to improve the performance of file transferring. By simultaneously downloading from multiple file servers, DFT is expected to increase throughput when the number of servers increases.

The experiment will test the file transfer rate when a DFT client is downloading a file from multiple servers. Different results will be compared when different numbers of file servers are chosen.

5.1.2 Experiment Environment

5.1.2.1 Network

DFT is designed to transfer long files over the Internet. Therefore, the network in the experiment will be over the Internet. The servers are distributed throughout the Internet and the DFT client that resides in the lab is connected to the Internet through the campus network.

5.1.2.2 Target File

The target file is referred to as F in the experiment. F's name is "Q3ADemo.exe", which is the demo version of Quake III. It is very popular and many FTP sites have copies of it. The length of F is of 47M bytes. This file is chosen because it is big enough to provide enough downloading time. During that time, the file transfer rate becomes stable and reliability tests can be performed. Yet, the size is not too big so that it represents a typical target file in today's Internet environment.

5.1.2.3 DFT Servers

In the experiment, seven servers contain F. They are listed in Table 6.

Server Address	Path
ftp.aros.net	/pub/games/Quake3/Q3ADemo.exe
ftp.cdrom.com	/.1/3dfiles/games/Q3ADemo.exe
ftp.pipex.net	/uunet/games/quake3/Q3ADemo.exe

ftp.u-net.net	/pub/games/quake3arena/Q3ADEMO/Q3ADemo.exe
ftp.cableinet.net	/pub/games/idsoftware/quake3/win32/old/Q3ADemo.exe
ftp.csufresno.edu	/pub/mirrors/q3demo/pc/Q3ADemo.exe
198.163.152.118	/home/jchen/Q3ADemo.exe

Table 6 DFT servers in experiment

Note that server 198.162.153.118 is a server in the lab. This server is used to simulate the failures.

5.1.2.4 DFT Client

The DFT client runs on a sun workstation in the lab. The client connects to the Internet through the campus network. The network interface of the client is a 10Mbps Ethernet. The Java environment is JDK2 sel.3. For the purpose of data analysis, the DFT client uses a consol mode instead of a graphical user interface mode. The consol mode uses a text interface and therefore is more efficient in terms of system resources.

5.2 Reliability Test

5.2.1 Server Failure Recovery

When the DFT client is downloading the target file, we reboot the server to simulate a server failure. The DFT client has a socket read timeout set to 20 seconds. When the server is down, no more data can be read on that socket. Twenty seconds later, the

timeout fires and the DFT client detects the failure. Therefore, the detecting time is set at 20 seconds. The experimental data shows it precisely. The timeout is set to 20 seconds so that a temporary delay on the network will not trigger switching of the server. In addition, when one downloading thread has failed, other downloading threads will still work. Therefore, during the time of failure recovery, the DFT keeps downloading the file. The overall downloading procedure will not be terminated.

The DFT client will try to reconnect to another file server after it detects the failure. This procedure depends on the response speed of that server. If the server is fast in response, the recovery time is short. Otherwise, the recovery time will be longer. It also depends on the amount of welcome messages that a server sends to a client when the client logs in. Our experiment shows a typical value of 3 seconds.

Table 7 shows the failure/recovery time of the experiment

Failed server	Alternative server	detecting time	recovery time	fail over time
198.162.153.118	ftp.aros.net	21 seconds	3 seconds	24 seconds
198.162.153.118	ftp.pipex.net	20 seconds	2 seconds	23 seconds

Table 7 Failure/recovery time when a server fails

5.2.2 Hardware Failure Recovery

This experiment simulates a link failure by unplugging the network cable of the server. DFT client use the same method to detect and recover link failure as server failure. Table 8 shows the test results of link failures.

Failed server	Alternative server	detecting time	recovery time	fail over time
198.162.153.118	ftp.aros.net	20 seconds	3 seconds	23 seconds
198.162.153.118	ftp.pipex.net	20 seconds	2 seconds	22 seconds

Table 8 Failure/recovery time when a server's link fails

5.2.3 DFT Client Failure Recovery

This experiment simulates the DFT client's failure by killing all DFT client processes on the workstation. Then, we execute the DFT client again and let it download the same file as before. The DFT client successfully checks the log file and resumes the interrupted downloading.

5.3 Performance Test

This experiment selects four servers to perform the performance test. We record the download speed of each thread associated with each server and plot them according to time. In order to get the speed of the transferring, we only use the stable section of the

speed to calculate the mean speed. Note that at the beginning, the speed increases sharply and becomes stable after about 50 seconds.

Figure 18, Figure 19, Figure 20 and Figure 21 illustrate the speed of each thread. Figure 18 corresponds to the server ftp.aros.net. Figure 19 illustrates the threads of the servers ftp.aros.net and ftp.cableinet.net. Figure 20 illustrates an additional server ftp.csufresno.edu and Figure 21 has an additional server ftp.cdrom.com. Table 9 gives the throughput comparison.

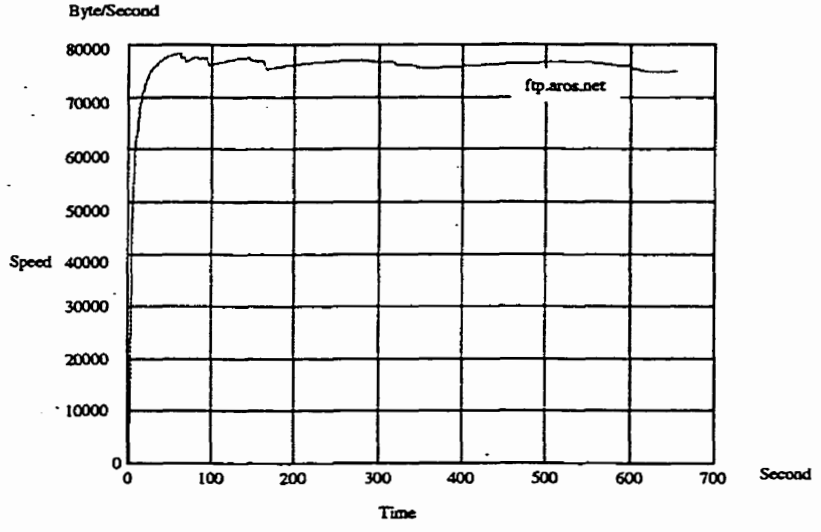

Figure 18 Downloading Speed of One Server

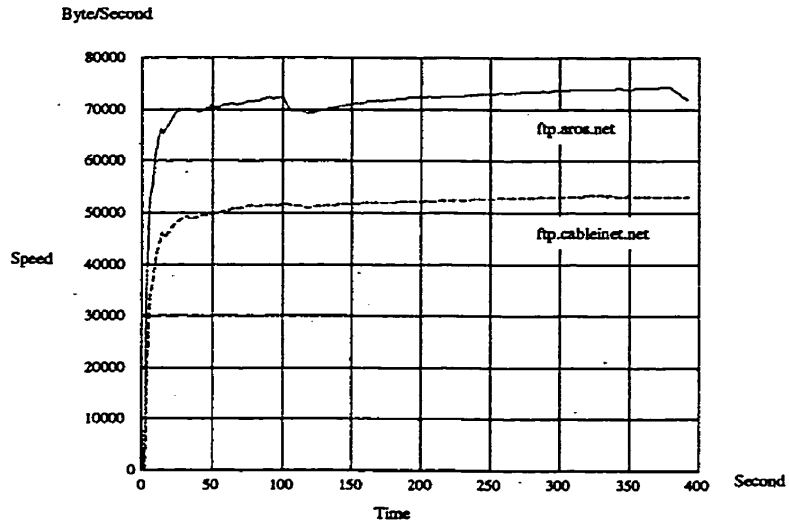

Figure 19 Downloading Speeds of Two Servers

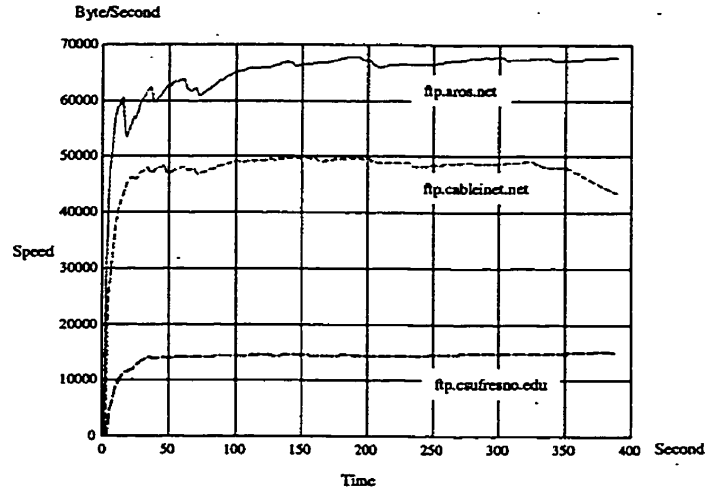

Figure 20 Downloading Speeds of Three Servers

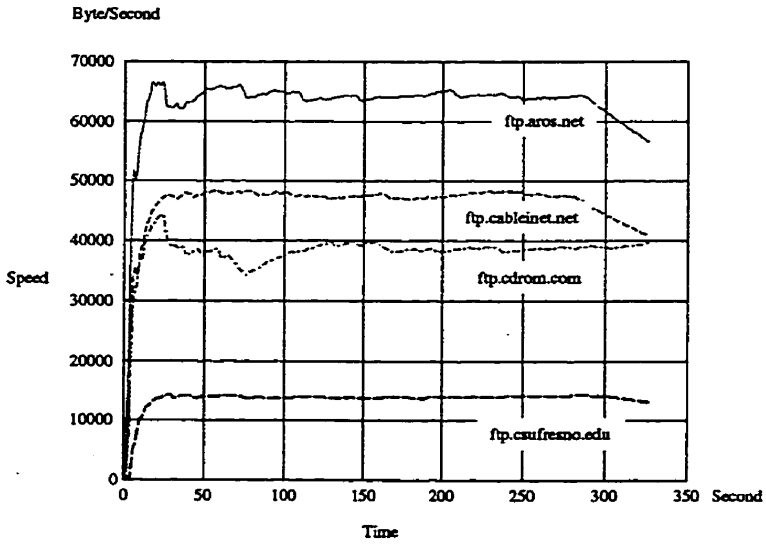

Figure 21 Downloading Speeds of Four Servers

Number of Servers	Throughput (Kbps)	Increasing percentage
1	610.3	0%
2	999.7	63.8%
3	1040.6	70.5%
4	1314.8	115.4%
5	1372.7	124.9%

Table 9 Downloading rate with different number of servers

The overall throughput of DFT is compared in Figure 22. In Figure 22, we found that generally speaking, downloading throughput increases when the number of servers increases. Specifically, the throughput of two servers is nearly the sum of each one's

throughput.

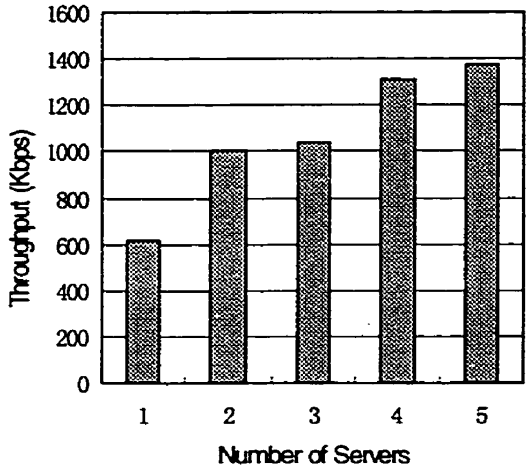

Figure 22 Throughput of DFT with Different Number of Servers

However, the throughput of five servers is very close to that of four servers. This may due to three reasons. First, the additional server's throughput is not significant. Second, the overall throughput reaches the limit of the network's bandwidth. Third, too many downloading threads increase the DFT client's burden. As the data shows in Table 9, the highest rate is near 1.4 M bps.

Summary

This chapter presented the design and implementation of the experiments and tests for the

DFT system. The main features of DFT, which are reliability and efficiency, are checked and tested. As the results show, DFT can detect and recover from server failures, network failures and the DFT client failures. By distributing the downloading task among multiple servers, DFT increases the overall throughput.

Chapter 6
Conclusions

This thesis presented the designed and implementation of a reliable file transfer mechanism over the Internet. It also presented a three layered failure model contributing to the failure recovery philosophy of DFT.

Distributed file transfer (DFT) increases the reliability of file transfer on the Internet. DFT achieves this goal by downloading different segments of a file from multiple file servers simultaneously. The reliability of DFT increases when the number of servers increases. DFT can be deployed in the Internet to provide a global level reliable file transfer system. The failure/recovery time of a single thread of DFT is adjustable. This thesis shows a typical failure/recovery time of around 20 seconds for the recovery of a hard failure.

The throughput of DFT increases when the number of servers increase. Due to the simultaneity feature of DFT, multiple downloading streams raise the throughput of DFT to the limit of the hardware. In the experiments of this thesis, the throughput of four servers increases by more than 100% than that of one server.

Future work of DFT includes increasing the capacity of the LDS. It is also better to implement an LDS search engine of its own and put its search results in dedicated database system. We may even consider increasing the reliability of the LDS. For example, we can provide multiple LDS for DFT client to increase reliability. On the server side, DFT can maintain some kind of agent like SNMP that reports the usage of the server, so that LDS can provide more precise information to the DFT client.

References

[Afreet] Afreet Software, Inc., http://www.netvampire.com

[AOM1998] M. Allman, S. Ostermann and C. Metz, "FTP Extensions for IPv6 and NATs", RFC 2428, Network Working Group, September 1998.

[APa] Arrowpoint Communication, Inc., "Arrowpoint Web Network Service – Enabling the Web-centric Internet", http://www.arrowpoint.com

[APb] Arrowpoint Communication, Inc., "A Comparative Analysis of Web Switching Architectures", http://www.arrowpoint.com

[Brisco1995] T. Brisco, "DNS Support for Load Balancing", RFC 1794, Network Working Group, April 1995.

[Comer1995] D. E. Comer, *Internetworking with TCP/IP Volume I: Principles, Protocols, and Architecture*, 3rd ed., Prentice Hall, 1995

[Couch1987] L. W. Couch II, *Digital and Analog Communication Systems*, 5th ed., Prentice-Hall International, Inc. 1987

[CS1994a] D. E. Comer, D L. Stevens, *Internetworking with TCP/IP Volume II: Design, Implementation, and Internals*, 2nd ed., Prentice Hall, 1994

[CS1994b] D. E. Comer, D. L. Stevens, *Internetworking with TCP/IP Volume III: Client-Server Programming and Applications*, 3rd ed., Prentice Hall, 1994

[CW1996] M. Campione, and K. Walrath, *The Java Language Tutorial: Object-Oriented Programming for the Internet*, Addison-Wesley, 1996

[DuaneC1996] D. C. Hanselmen, , *Mastering MATLAB: A Comprehensive Tutorial and Reference*, Prentice Hall, 1996

[F5] F5 Networks, Inc, http://www.bigip.com

[Flanagan1996] D. Flanagan, *Java in a Nutshell*, O'Reilly & Associates, Inc., 1996

[Harold1997] E. R. Harold, *Java Network Programming*, O'Reilly &Associates, Inc., 1997

[Headlight] Headlight Software, http://www.getright.com

[HL1997] M. Horowitz and S. Lunt, "FTP Security Extensions", RFC 2228, Network Working Group, October 1997.

[ISC2000] Internet Software Consortium, http://www.isc.org/ds/hosts.html, Internet Software Consortium, 2000

[JM1997] J. R. Jackson and Alan L. McClellan, *JAVA by Example*, Prentice-Hall, 1997

[Linden1996] P. Van Der Linden, *Just Java*, Englewood Cliffs, Prentice Hall, 1996

[Lycos] Lycos, Inc., http://ftpsearch.lycos.com

[Mullender1993] S.J. Mullender, *Distributed Systems*, 2nd ed., New York, ACM Press 1993

[Paxon1994] V. Paxson, "Growth Trends in Wide-Area TCP Connections". *IEEE Network*, Vol. 8 No. 4, pp. 8-17, July 1994.

[PR1985] J. Postel, J. Reynolds, "File Transfer Protocol (FTP)", RFC 959, Network Working Group, October 1985.

[Radiate] Radiate, Inc., http://www.gozilla.com

[Ryan1999] J. Ryan, *Designing and Implementing a Virtual Private Network*, the Applied Technologies Group, Inc., 1999

[Schach1998] S. R. Schach, *Software Engineering with JAVA*, 4th ed., McGraw-Hill, 1998

[Speedbit] Speedbit, http://www.speedbit.com

[Stallings1994] W. Stallings, *Data and Computer Communications*, 4th ed., Macmillan Publishing Company, 1994

[Stevens1998] W. R. Stevens, *Unix Network Programming*, Volume 1, 2nd ed. Prentice Hall, 1998.

[Tanenbaum1996] A. S. Tanenbaum, *Computer Networks*, 3rd ed., Prentice Hall, 1996.

Appendix A
Numeric Order List of FTP Reply Codes

110 Restart marker reply.

In this case, the text is exact and not left to the particular implementation; it must read:

MARK yyyy = mmmm

Where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").

120 Service ready in nnn minutes.

125 Data connection already open; transfer starting.

150 File status okay; about to open data connection.

200 Command okay.

202 Command not implemented, superfluous at this site.

211 System status, or system help reply.

212 Directory status.

213 File status.

214 Help message.

On how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.

215 NAME system type.

Where NAME is an official system name from the list in the Assigned Numbers document.

220 Service ready for new user.

221 Service closing control connection.

Logged out if appropriate.

225 Data connection open; no transfer in progress.

226 Closing data connection.

Requested file action successful (for example, file transfer or file abort).

227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

230 User logged in, proceed.

250 Requested file action okay, completed.

257 "PATHNAME" created.

331 User name okay, need password.

332 Need account for login.

350 Requested file action pending further information.

421 Service not available, closing control connection.

This may be a reply to any command if the service knows it must shut down.

425 Can't open data connection.

426 Connection closed; transfer aborted.

450 Requested file action not taken.

File unavailable (e.g., file busy).

451 Requested action aborted: local error in processing.

452 Requested action not taken.

Insufficient storage space in system.

500 Syntax error, command unrecognized.

This may include errors such as command line too long.

501 Syntax error in parameters or arguments.

502 Command not implemented.

503 Bad sequence of commands.

504 Command not implemented for that parameter.

530 Not logged in.

532 Need account for storing files.

550 Requested action not taken.

File unavailable (e.g., file not found, no access).

551 Requested action aborted: page type unknown.

552 Requested file action aborted.

Exceeded storage allocation (for current directory or dataset).

553 Requested action not taken. File name not allowed.