Improved Distributed File Transfer (DFT) on Internet

By

Yuhong Li

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

Improved Distributed File Transfer (DFT) on Internet

BY

Yuhong Li

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

YUHONG LI ©2005

To my beloved parents.

Abstract

File transfers happen every moment on the Internet. The File Transfer protocol (FTP) is one of the most commonly used protocols on the Internet. FTP has some limits and problems such as being server-centered and single-threaded. Server-centered characteristics can easily degrade the performance of the server and in the worst case crash the server. Nowadays there are usually multiple copies of a file scattered on the Internet. Using traditional FTP, some of them are very busy and the others may be very quiet. In addition, single threaded downloading is very inefficient. Improving the reliability and efficiency of FTP is the motivation and goals of this thesis.

To achieve this goal, a mechanism called Distributed File Transfer (DFT) is designed and implemented in this thesis. In DFT, multiple servers holding the same copy of files are used as FTP servers. The client connects to a Load Distributing Server (LDS) to get FTP server information by an Internet search or through a local database. Connections are made to these FTP servers and the file is downloaded in parallel. The user uses multiple threads to download from each of the servers. This overcomes server-centered FTP drawbacks. The user can still continue the download from the other servers even when some of them experience bottlenecks or go down. The experiments done on the files with size less than 1 GB shows the reliability and efficiency trends when using different server combination and using different number threads from each server.

Acknowledgements

During the days I was working on this thesis, many people have contributed to it. First of all, I thank my advisor, Dr. Robert D. McLeod, for his guidance and advice on this thesis and my study through these years. I really appreciate his kindness and all the help that he has given to me.

Thanks to Dr. Rasit Eskicioglu and Dr. Ekram Hossain for taking the time to be on my thesis committee for reading my thesis and for their efforts.

I want to thank my parents, brothers and husband for their persistent encouragement, inspiration and support.

I would also like to thank all the people whose names are not mentioned here but provided me with all the encouragement and support.

Contents

Abstract iii

Acknowledgements iv

Contents v

Figures viii

Tables x

Chapter 1: Introduction 1

1.1. The Internet 1
1.2. TCP/IP 3
1.3. The Creation of Java 5
1.4. Understanding FTP 6
1.4.1. FTP Protocol 6
1.4.2. A Sample FTP Session 7
1.4.3. FTP Common Commands 10
1.4.4. File Transfer Resume 13
1.4.5. Active FTP vs. Passive FTP 14
1.4.5.1. Active FTP 14
1.4.5.2. Passive FTP 16
1.4.6. The Problems of FTP 18
1.5. Exploration on the Improvement of File Download 19
1.5.1. One Server with One Thread Model 20
1.5.2. Improved One Server with Multiple Threads Model 20
1.5.3. Peer-to-Peer File-Sharing 21
1.5.4. Distributed File Download Model 21
1.5.5. Improved Distributed File Download Model 22
1.6. Digital Signatures 23
1.6.1. Introduction to Digital Signature 23
1.6.2. Message Digests and Digest Algorithms 24
1.6.3. Signature Algorithms 24
1.6.4. Digital Signatures in Java 26
1.7. Summary 27

Chapter 2: DFT System Objectives and Requirements 28

Chapter 3: DFT System Architecture 30

3.1. System Architecture Diagram 30
3.2. Distributed Architecture of DFT 31
3.3. Distributed File Transfer (DFT) Client 32
3.4. Two-Layer Multi-Thread Parallel Download 33
3.5. Load-Distributing Server (LDS) Server 36
3.6. DFT File Query and Response Model 37
3.7. DFT Two-Layer Control Connection Model 39
3.8. DFT Data Connection Model 43

3.9. Failure Detection 43
3.9.1. Two kinds of failures 44
3.9.1.1. Network failures *44*
3.9.1.2. Application failures *44*
3.9.2. Two kinds of failure detection methods 45
3.9.2.1. Timeout detection *45*
3.9.2.2. Application failure code detection *46*
3.10. Failure Recovery 46
3.11. Checkpoint Resume 47
3.12. Local Directory 48
3.13. Digital Signature 48
3.14. Error Handling 49
3.15. Summary 50

Chapter 4: Implementation 51

4.1. Implementation Tools 51
4.1.1. Java Programming Language 51
4.1.2. Java Foundation Classes (JFC) 53
4.1.3. JCreator Integrated Development Environment 54
4.2. DFT Package Implementation 55
4.2.1. DFT Package Directory Structure 55
4.2.2. DFT Packages 56
4.2.2.1. Package dft.gui *56*
4.2.2.2. Package dft.client *56*
4.2.2.3. Package dft.server *57*
4.2.2.4. Package dft.util *58*
4.2.2.5. Package dft.gensig *58*
4.2.2.6. Package dft.versig *58*
4.3. DFT Modules Implementation 58
4.3.1. DFT Client Side Implementation 59
4.3.1.1. DFT Client Workflow *59*
4.3.1.2. Download Process Implementation *59*
4.3.1.3. Download Thread Implementation *62*
4.3.1.4. Sub-Download Thread Implementation *63*
4.3.1.5. DataConnection Implementation *64*
4.3.1.6. ControlConnection Implementation *65*
4.3.1.7. Timer Implementation *66*
4.3.1.8. check() Implementation *67*
4.3.1.9. LogTimer Implementation *68*
4.3.1.10. Sub-Download SpeedTimer Thread Implementation *69*
4.3.2. DFT Server Side Implementation 70
4.3.2.1. LDS Server Implementation *70*
4.3.2.2. Search Engine Implementation *71*
4.4. DFT System Module List 73
4.5. Summary 74

Chapter 5: Experiment and Data Analysis 75

5.1. Experiment Goals and Design 75
5.1.1. Experiment Goals: 75
5.1.2. Experiment Design: 76
5.1.2.1. Reliability Test: *76*
5.1.2.2. Digital Signature Generation and Verification Test *77*
5.1.2.3. Efficiency Test *77*
5.1.2.4. Auto Optimization Test *77*

5.2. Experiment Environment 77
5.2.1. Networks 78
5.2.2. Downloaded Files 79
5.2.3. FTP Servers: 80
5.2.4. LDS Server and DFT Client: 81
5.3. Reliability test 82
5.3.1. Server Failure Recovery 82
5.3.2. Network Failure 84
5.4. Signing a File and Verify a Signature 84
5.4.1. Signing a File 84
5.4.2. Verifying a Signature 86
5.5. Efficiency Test on Files Less Than 100 MB 87
5.5.1. Test Results 88
5.5.2. Experiment Analysis 91
5.5.2.1. Single server Download – one vs. multiple threads 91
5.5.2.2. Multiple servers Download – one thread vs. multiple threads 92
5.5.2.3. One Server vs. Multiple Servers 93
5.5.2.4. Elements Affecting Download Speed 94
5.5.2.5. Low Speed Server Switch 95
5.6. Efficiency Test on Bigger Files 95
5.6.1. Test Results 95
5.6.2. Experiment Analysis 96
5.7. Summary 97

Chapter 6: Conclusions and Future Work 98

References 101

Appendix 104

Figures

Figure 1.1 Internet growth trends 2
Figure 1.4.5.1 Active FTP Client and Server Connection 15
Figure 1.4.5.2 Passive FTP Client and Server Connection 17
Figure 1.5.1 One Server with One Thread 20
Figure 1.5.2 One Server with Multiple Threads 20
Figure 1.5.3 P2P File Sharing and Download 21
Figure 1.5.4 Multiple Servers with One Thread 22
Figure 1.5.5 Multiple Servers with Multiple Threads Model 22
Figure 1.6.3 Producing and Verifying a DSA Signature 26
Figure 3.1 DFT System Architecture Diagram 30
Figure 3.4a DFT Multi-Thread Parallel Processing Diagram 33
Figure 3.4b DFT Multi-Thread Parallel Processing Diagram 35
Figure 3.6 DFT File Query and Response Model Diagram 37
Figure 3.7 DFT Two- layer Control Connection Model 40
Figure 4.3.1.1 DFT Client Side Workflow 59
Figure 4.3.1.2a Download Process Workflow (1) 60
Figure 4.3.1.2b Download Process Workflow (2) 61
Figure 4.3.1.3 Download Thread Workflow 62
Figure 4.3.1.4 Sub-Download Thread Workflow 63
Figure 4.3.1.5 DataConnection Workflow 64
Figure 4.3.1.6 ControlConnection Workflow 65
Figure 4.3.1.7 Timer Workflow 66
Figure 4.3.1.9 LogTimer Workflow 68
Figure 4.3.1.10 Sub-Download SpeedTimer Thread Workflow 69
Figure 4.3.2.1 LDS Server Workflow 70
Figure 4.3.2.2 FTP Search Engine Adaptor Workflow 72
Figure 5.2.1 DTF System Test Networks 78
Figure 5.3.1a Pick up the internal server and an external one 82
Figure 5.3.1b internal server Switched to the external Server 83
Figure 5.4.1a Find and choose the file being signed on 85
Figure 5.4.1b Successful signature generation message 85
Figure 5.4.1c Key and signature files are generated 86
Figure 5.4.2a Looks in the folder where the 3 files are 87
Figure 5.4.2b Signature is verified successfully 87

Figure 5.4.2c Fail to verify the signature *87*
Figure 5.5.2.2 Duration vs. Thread Numbers *92*
Figure 5.5.2.3 Download Time vs. Server Numbers *94*

Tables

Table 1.2 OSI 7 Layers and TCP/IP Suite *4*
Table 4.4: System Unit Module List Table *73*
Table 5.2.3: FTP Servers Used in the Experiments and Tests *81*
Table 5.3.1: Results of Two Tests *83*
Table 5.5.1a: Testing Data on Downloading mysql-4.1.12a-win32.zip *88*
Table 5.5.1b: Testing Data on Downloading OOo_1.1.4_Win32Intel_install.zip *90*
Table 5.6.1: Testing Data on Downloading Big Files *95*

Chapter 1: Introduction

1.1. The Internet

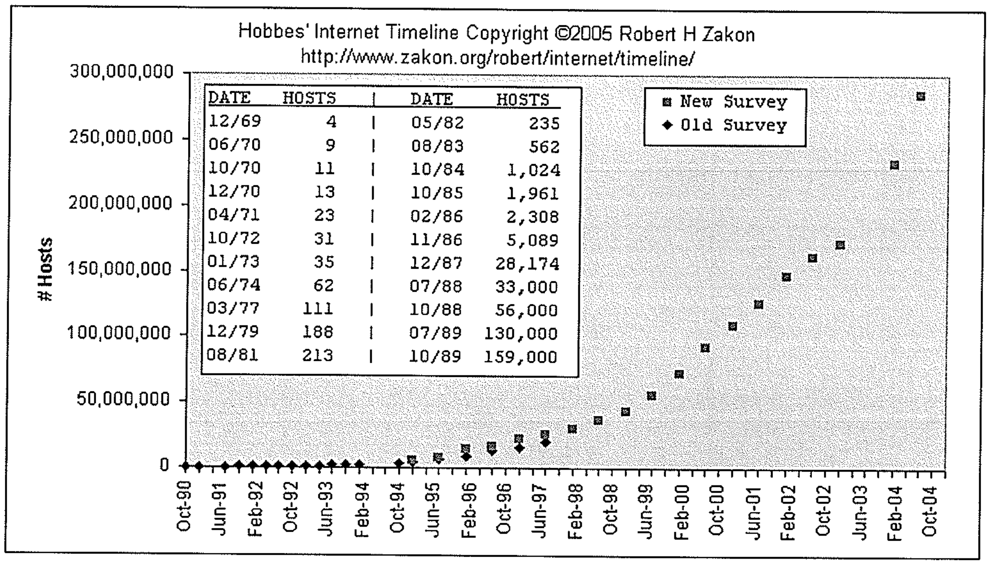
The Internet started from the ARPANET back in the late 1960s in the United States. ARPANET was sponsored by the US Defense Advanced Research Projects Administration (DARPA). It was designed and implemented as a decentralized packet-switching network that could create a way for network communications to occur between two systems in such a way that reliance on a single link wasn't required. In other words the network communication system could find alternative paths through complex matrices of wires if some or most of the wires were broken during an attack.

The protocol used to communicate between hosts on the ARPANET at its early stage was the Network Control Protocol (NCP), which enabled hosts running on the same network to transfer data. In 1973, development began on a protocol suite now known as the Transmission Control Protocol/Internet Protocol (TCP/IP). The major goal of this protocol was to enable separate computer networks to interconnect and communicate with one another.

In 1982, the term 'internet' was defined as a connected set of separate networks using the TCP/IP protocol suite. The 'Internet' was defined as connected TCP/IP internets. In 1983, the ARPANET changed its core networking protocols from NCP to TCP/IP officially, marking the start of the Internet as we know it today.

By this time, ARPAnet was connecting machines across the continent (US) with an estimated rate of connection of one new machine every 20-days [1].

With the technology advances such as the NSFNET, T1, Asynchronous Transmission Mode (ATM) and the enhancement of the TCP/IP protocol suite, the Internet's pace of growth has been spectacular since the early 1990s. In 1992, the nodes on the Internet counted 1 million; in January 2001, the number reached 100 million; in September 2002, the number jumped to 200 million and in January 2005, the number reached to 350 million. By September 2002, there were 840 million people using the Internet from more than 218 of 246 countries on the earth. Figure 1.1 shows the Internet growth trends indicated by the host count [2].



DATE	HOSTS	DATE	HOSTS
12/69	4	05/82	235
06/70	9	08/83	562
10/70	11	10/84	1,024
12/70	13	10/85	1,961
04/71	23	02/86	2,308
10/72	31	11/86	5,089
01/73	35	12/87	28,174
06/74	62	07/88	33,000
03/77	111	10/88	56,000
12/79	188	07/89	130,000
08/81	213	10/89	159,000

FIGURE 1.1 INTERNET GROWTH TRENDS

The Internet began as a network for physicists and researchers and has evolved into a network for all kinds of people around the globe. The growth is not simply a matter of the increase in the number of new members who join the Internet community, but the manner in which the Internet continues to invade every aspect of the modern life, reshaping business priorities, consumer requirements, and general commercial attitudes.

File transfer over the Internet has been a big contributor to the growth of Internet. This thesis provides a design and implementation on how to transfer files over the Internet in a more reliable and effective way.

1.2. TCP/IP

A protocol is an agreement used for communication between two networked computers. It defines how data should be packaged for transmission on the network so the receiving host can unpackage it on the reception. For two hosts to communicate on a network, the hosts must be using the same protocol. As mentioned above, TCP/IP replaced NCP in 1983 beginning the new era of the Internet. Because of the success of the Internet, TCP/IP has become the standard of today's network: the Internet, local and wide area networks. TCP/IP is the most widely used protocol. The TCP/IP protocol establishes the technical foundation of the Internet.

The Transmission Control Protocol and Internet Protocol (TCP/IP) are the two protocols in the TCP/IP protocol suite. Internet Protocol (IP) specifies how data is routed from one computer to another. The Transmission Control Protocol (TCP) verifies whether or not the information arrived at the designated computer and if not, makes sure that the information is sent again. So the protocols of TCP/IP define the network communication process and, more importantly, define how a unit of data should look and what information it should contain so that a receiving computer can interpret the message correctly. TCP/IP and its related protocols form a complete system defining how data should be processed, transmitted, and received on a TCP/IP network.

The protocols in the TCP/IP suite function primarily in the Network layer (layer 3), Transport layer (layer 4) and Application layer (layer 7) based on the OSI network reference model. TCP/IP supports all popular layer 2 protocols as well. The applications in the TCP/IP suite are normally operating directly on top of the Transport layer protocol TCP or UDP without the support of the Presentation layer (layer 6) and Session layer (layer 5). Table 1.2 shows the OSI 7 layers and TCP/IP suite.

TABLE 1.2 OSI 7 LAYERS AND TCP/IP SUITE

Layer	**TCP/IP Suite**
7 – Application	HTTP, SMTP, SNMP, FTP, Telnet, NFS, NTP
6 - Presentation	XDR, SSL, TLS
5 – Session	Session establishment for TCP
4 – Transport	TCP, UDP, RTP, SCTP
3 – Network	IP, ICMP, IPsec, ARP, RIP, OSPF, BGP
2 - Data Link	ARP, IARP, RARP, SLIP
1 – Physical	

1.3. The Creation of Java

The explosive growth of the Internet has dramatically transformed not only the way people do business and get entertainment, but it also has forced programmers to think about programs in new ways. Since networks consist of many different kinds and sizes of computers, all information and programs on the Internet must be usable without modification due to the variety of computers. There was a need to write programs that can run on any of these machines so that the look and feel doesn't change substantially across computers running different operating systems (platform independence).

Java arose as a new programming language under these circumstances. Java evolved from Oak, a language developed by Sun Microsystems in the early 1990s. Oak was intended to be a platform independent language for use in consumer electronic devices (for example the handheld devices and set-top boxes). However, Oak was unsuccessful in its initial intension.

It was the advent of the World Wide Web (www) that propelled Java into prominence. With people running different operating systems and wishing to access programs available on the Internet, platform independent programming became very important. Sun realized that they had been working on a programming language that had the capability to embed intelligent, interactive content into a web page. The focus of the language development changed from consumer electronics to Internet programming. In 1995, Sun changed the name

Oak to Java. Java has taken the software community by storm and achieved phenomenal success.

Java provides developers with many features such as object-orientation, simplicity, robustness, security, multi-thread, platform independence etc. While most of these are present in other languages, Java combines all of these together into one language [3]. Java is a programming language expressly designed for use in the distributed environment of the Internet. It has a rich library for network programming. In this thesis, Java was chosen as the programming language to implement the system functions. More detailed information about Java will be given in the implementation section.

1.4. Understanding FTP

1.4.1. FTP Protocol

File Transfer Protocol, also known as FTP, is the protocol for exchanging files over the Internet. FTP works in the same way as HTTP for transferring Web pages from a server to a user's browser and SMTP for transferring electronic mail across the Internet in that, like these technologies, FTP uses the TCP/IP protocols to enable data transfer. FTP is most commonly used to download a file from a server or to upload a file to a server through the Internet.

An FTP server is the site where the user logs in and downloads files from. An FTP client is the software the user uses to download files with the FTP client installed on the user's machine. The following sections describe how FTP works.

1.4.2. A Sample FTP Session

The client program connects to a FTP server on the network. Once connected, the FTP server sends a welcome message to the client over the open socket (network) connection.

```
Server: 220 Sample FTP server ready. Please give user-name

Client: USER anonymous

Server: 331 User name OK. Please give your email address as
password

Client: PASS joe@nowhere.comm

Server: 230 User logged in
```

From the above, the client and server are communicating in plain text. The digits in the server replies are 'reply-codes' defined by the FTP protocol. The uppercase words in the beginning of the client commands are command verbs that also are defined by RFC 959 [4]. The protocol is designed in a way that makes it easy for machines and humans to understand the dialog. In most cases, the client programs don't have to interpret the text after the reply code.

If the user wants to see the available files and directories, they would issue a LIST command in the client program.

```
Client: TYPE A

Server: 200 Type set to A

Client: PASV

Server: 227 Entering passive mode (130,179,16,12,28,46)

Client: LIST

Server: 150 Opening ASCII mode data connection for /bin/ls

Server: 226 Transfer complete
```

The command 'TYPE A' tells the server to send the directory/file listing as plain ASCII.

The command 'PASV' tells the server to prepare for a new socket connection by creating a new socket and listens for a connection from the client. Now, things get a little more complicated. The server reply includes an IP address and a port number, encoded as 6 digits, separated by commas. The client must find and understand this address in order to receive the listing.

The LIST command tells the server to give a directory/file listing. Now the server replies with two lines. The first line tells the client that the listing is ready, and the client can go on and make a new connection to the server. The client connects to the IP address given by the PASV reply, and receives data until there is no more data to get. Then it closes the temporary data connection and switches back to the control connection to get the second reply line, which tells if the server has transferred the whole listing.

In order to receive a directory listing, the client and server use two socket connections, one for the control flow (client sends commands, the server replies in plain text) and one for the data connection (which is continuous and goes in one direction only). Next time a directory-listing is sent, the server and client will use another new (temporary) socket connection for the transfer.

When the users find an interesting file, they give the FTP server the command to get it.

```
Client: TYPE I
Server: 200 Type set to I
Client: PASV
Server: 227 Entering passive mode (130,179,16,12,28,46)
Client: RETR test.zip
Server: 150 Opening BINARY mode data connection for test.zip
Server: 226 Transfer complete
```

As you see, the server and client use the exact same method to get a file as to get a directory listing. The only change is that the RETR command is used instead of the LIST command. In this case the file was a .zip archive which was in binary format, and since such files can't be translated to text, the FTP client switched to binary mode (TYPE I). Files and directory listings can be transferred in both binary and text-mode.

It is easy to connect to an FTP server with a telnet client and give commands, but due to the fact that the file-transfers use a separate socket connection, it is not easy to transfer files without an FTP client.

1.4.3. FTP Common Commands

The FTP commands listed in this section are the typical FTP commands. Each one of these commands is commonly used by FTP clients and should be supported in some fashion by all FTP servers.

USER (USER NAME)

All FTP communications begin with the USER command. This command takes a single argument: the username that the client wishes to be authenticated with. In a Windows NT environment, this may include both the domain name and the username. For example, when logging onto an NT server that is a member of a domain, the client may transmit the command:

```
USER domain\username
```

The most common argument to the USER command is anonymous. Anonymous logons to FTP are common on the Internet, where a large percentage of FTP servers carry information for the general public.

PASS (PASSWORD)

The PASS is generally the second command transmitted by a client to the server. It carries an argument the password for the user already specified by the USER command. This command is as simple as it seems: There is no encoding or encryption of the password, it is simply clear text. An example of transmitting a password from a client to a server:

```
PASS password
```

CD (change working directory)

The CD command changes the directory the FTP server is working with. The only argument for this command is the new directory, in either absolute form or relative form. Examples of both of these forms are given here:

```
cd /usr/root

cd /usr/root/

cd documents
```

The first command changes the current directory to /usr/root, regardless of what the current directory is. The second command illustrates an optional slash at the end of the directory name. The third command moves into a child directory of the current directory named documents; it only works from directories that have a subdirectory called documents.

LS (DIRECTORY LISTING)

The LIST command causes a data transfer to occur that will contain the directory listing for the current working directory. An absolute or relative path can be given as an optional argument if a directory listing for another directory is desired.

RETR (DOWNLOAD)

When the RETR command is issued from the client, a data transfer connection is established. The RETR command takes as an argument the path to the file to be transferred. For example, to use the RETR command to transfer the file */documents/file.html*, issue the command:

```
RETR /documents/file.html
```

STOR (UPLOAD)

Similar in function and execution to the RETR command, the STOR command sends a file from the client to the server. The only argument for the STOR command is the destination location on the server. If the file already exists in the destination directory, it is automatically overwritten. To upload the file file.html to the /documents directory, issue the command:

```
STOR /documents/file.html
```

REST (RESTART)

The REST command is used to continue a session that has been interrupted. The REST command has an argument, an integer that represents the position in the file where transfer should begin. For example, to restart a transfer at byte 4096 in a file, the client would issue the following commands to the server:

```
REST 4096
RETR /documents/file.zip
```

It is important to understand that the command that follows the REST command must be a transfer of some kind, either a STOR or RECV.

QUIT (LOG OUT)

The QUIT command is sent to the server to indicate that the FTP session is over. This command takes no arguments.

1.4.4. File Transfer Resume

FTP can resume the interrupted file transfer session by sending a REST command to FTP Server. REST command takes a parameter as the offset where transfer should resume.

```
REST <offset>
RETR <file path and name>
```

These two commands will set the checkpoint <*offset*> first and then download the file <*file path and name*> from that point on.

1.4.5. Active FTP vs. Passive FTP

FTP is a peculiar protocol because it uses two sockets. The main socket (TCP port 21) handles the commands the FTP client sends to the server as well as the associated response from the server. The other port, which is at port 20 by default, handles data. Depending on the transfer mode used, the data port is not always on port 20. It can be a different port number.

Active and passive are two kinds of FTP transfer modes. It is necessary to differentiate active and passive mode FTP to deal with firewalls and other Internet connectivity. For example, when the DFT client was first developed, it ran well with no firewalls set up on the client. But when run from within a firewall, a problem was encountered that the outside FTP server cannot connect to the local port of the client machine. The reason was that all the server's attempts to connect to the client machine on a random unprivileged port are blocked by the firewall. The solution to this problem was using passive FTP connection.

1.4.5.1. Active FTP

In the active mode FTP, the client connects from a random unprivileged port N (N > 1024) to the FTP server's command port, port 21. Then, the client starts listening on port N+1 and sends the FTP command *PORT N+1* to the FTP server. The server will then connect back to the client's specified data port from its local data port.

In the eyes of the server side firewall, the connections from the client to its command port, port 21 and to its data port, port 20 are regarded as normal connections and thus neither will be blocked. But in the eyes of the client side firewall, the connections from the server are not to its ports, port 21 and port 20, rather to its random unprivileged ports, port N (N > 1024) and N+1. These connections are usually considered as something that should be blocked. Figure 1.4.5.1 depicts the connections in active FTP mode:

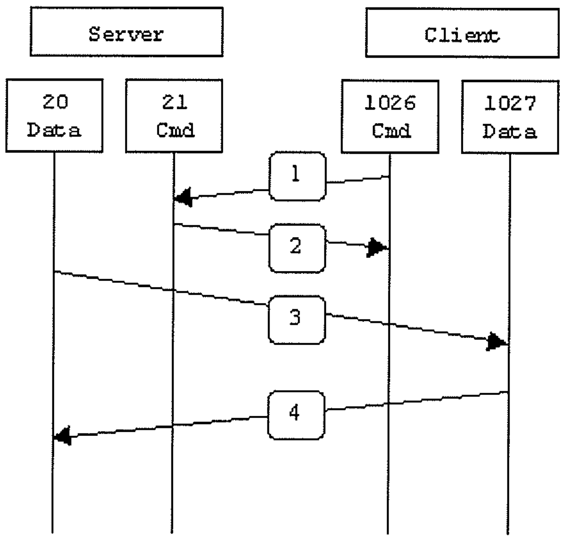


FIGURE 1.4.5.1 ACTIVE FTP CLIENT AND SERVER CONNECTION

In the above figure, the communication steps are:

- Step 1: The client connects from port 1026 (a random unprivileged port N) to FTP server's command port 21, and sends FTP command PORT 1027 (N+1) to the FTP server, then listens to port 1027 (N+1)
- Step 2: The FTP server sends back ACK command to the client

- Step 3: The server initiates connection from data port 20 to the client's specified port 1027
- Step 4: The client sends back ACK command to server

In the active mode FTP, the FTP client doesn't make the actual connection to the data port of the server. It merely tells the server what port it is listening on and the server connects back to the specified port on the client. It is the server that initializes a connection to the client. If the client is behind firewall, the server's attempt will be blocked by the firewall. So the active mode FTP is beneficial to the FTP server, but causes problems for the client.

1.4.5.2. Passive FTP

The passive mode (PASV) was developed to solve the problem of active mode FTP. In passive mode, the client initiates both connections to the server, thus solves the problem that the client's firewall filters the incoming data port connection from the server.

In passive mode, the client first opens two random unprivileged ports locally (N > 1024 and N+1). The port N contacts the server on its command port, port 21, but instead of then issuing a PORT command and allowing the server to connect back to its data port, the client issues the *PASV* command. The server then opens a random unprivileged port (P > 1024) and sends the *PORT P* command back to the client. The client then initiates the connection from port N+1 to port P on the server to transfer data.

Since both connections are initialized by the client, there is no incoming connections being considered as abnormal and blocked. In the case of the server-side firewall, the control connection from the client to its command port, port 21 is normal. But the data connection to its data port, port P which is a random unprivileged will be blocked.

Figure 1.4.5.2 depicts how the communication process happens between client and server in passive mode.

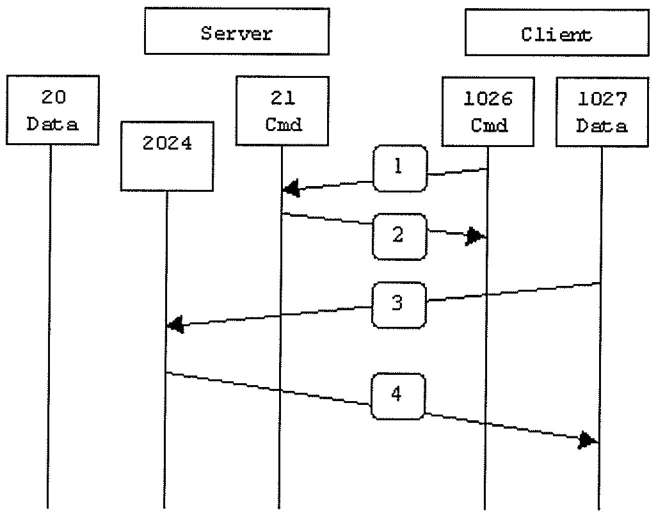


FIGURE 1.4.5.2 PASSIVE FTP CLIENT AND SERVER CONNECTION

- Step 1: The client opens two unprivileged ports: 1026 and 1027; 1026 connects to the server's port 21, and sends command *PASV* to server

- Step 2: The server opens an unprivileged port 2024 and sends command *PORT 2024* back to client which tells the client what port it is listening on for the data connection.
- Step 3: The client initiates the data connection from port 1027 to the specified server data port 2024
- Step 4: The server sends back command ACK to the client's data port 1027.

The passive mode relieves the problem from the client, but causes a server side problem. Fortunately, many FTP servers allow the administrator to specify a range of ports which the FTP server will use and thus allows remote clients to make connections without being blocked.

Since the DFT system runs behind firewalls, it will use the passive FTP mode.

1.4.6. The Problems of FTP

Though FTP is widely used and easy to implement, it also faces problems with reliability, performance and scalability. FTP is typically a single-threaded, server-centered downloading mechanism which means a server hosts a file, and clients connect to the server for downloading the file. Problems arising include:

- When multiple clients connect to the server at the same time, the load can easily exceed the capacity of the server and cause server congestion or a server crash.

- When the server is brought down for upgrade or maintenance, all users will lose their connections with the server and the FTP sessions will be interrupted.
- With the growth of the Internet, it is common that the same copy of file is mirrored on different servers scattered on the Internet. Since people like to download files from servers they are familiar with, the load may not be distributed among the main server and its mirror sites.

Looking into the problems, if there is a way to improve the mechanism of traffic allocations, the problems that FTP faces can be solved. If the user can download a file from multiple servers that are holding the same copy of file (mirrors), when one or some of servers are down or busy, the user can still download the file without being aware of server failure. The users will not rely on any one server. Resolving FTP problem is the motivation of my thesis.

1.5. Exploration on the Improvement of File Download

With the growth of Internet, exploring how to improve the reliability and efficiency of file download has never stopped. Technologies and software for file transfer can be grouped into two categories: server-centered download and server-scattered download. The difference is whether the download is from a single server or from multiple scattered servers.

1.5.1. One Server with One Thread Model

This is the traditional model. All downloads are from a central server each with one thread (Figure 1.5.1). The classic command line FTP clients under Windows and Linux behave in this way.

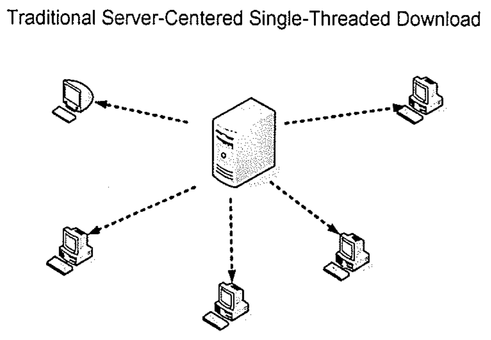


FIGURE 1.5.1 ONE SERVER WITH ONE THREAD

1.5.2. Improved One Server with Multiple Threads Model

This is an improvement for the one server with one thread model (Figure 1.5.2). In this model, the download is still from a central server but with multiple threads. There are many download managers such as FlashGet [5] and Internet Download Manager [6] that work in this way [7].

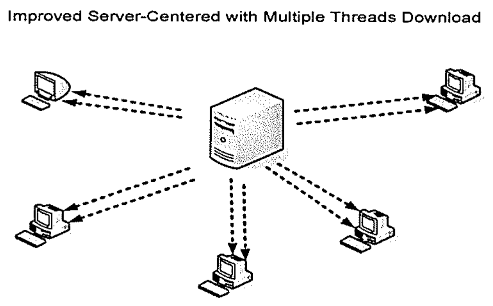


FIGURE 1.5.2 ONE SERVER WITH MULTIPLE THREADS

1.5.3. Peer-to-Peer File-Sharing

Figure 1.5.3 is a typical peer-to-peer file-sharing protocol (P2P) download model [9]. In this model, direct connections are set up in between users. Each user lets other users download from their computer while they are downloading from other users' computers. They behave as both client and sort of a server at the same time. But this 'server' is not a dedicated server. When the P2P client shuts down, this server disappears. Thus this kind of server's availability is not stable. eDonkey [9] and Bittorrent [10] are two of the top P2P software that provide this kind of file sharing and download functionality.

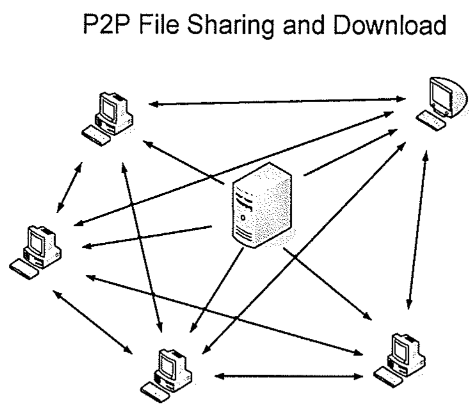


FIGURE 1.5.3 P2P FILE SHARING AND DOWNLOAD

1.5.4. Distributed File Download Model

This is a distributed file transfer (DFT) (Figure 1.5.4). Downloads are from multiple servers instead of a single central server. The servers are dedicated which are not like the 'servers' in P2P file-sharing model. The user sets up one connection to each server. Compared to the server-centric download, this improves the availability, reliability and efficiency of FTP downloads. The download manager GetRight [11] has this functionality. Xin Fang described a mechanism of this model in her thesis [12].

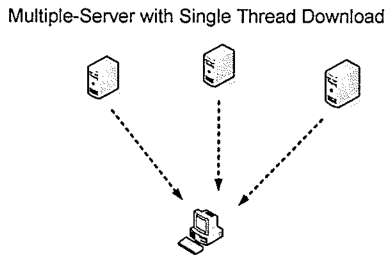


FIGURE 1.5.4 MULTIPLE SERVERS WITH ONE THREAD

1.5.5. Improved Distributed File Download Model

This is the improvement of the above DFT model (Figure 1.5.5). In this distributed download model, the user can download from multiple servers and establish multiple connections with each server. Its mechanism, functionalities, architecture and implementation will be described in details in the following chapters.

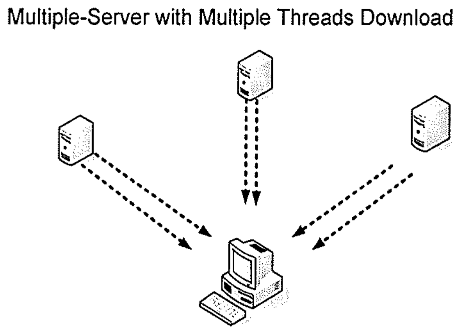


FIGURE 1.5.5 MULTIPLE SERVERS WITH MULTIPLE THREADS MODEL

This thesis focuses on improving the efficiency of transferring files of size less than 1Gigbyte.

1.6. Digital Signatures

One big concern in the file transfer on the Internet is how to make sure the transferred file is from the purported sender and has not been altered during the transit. Digital signature addresses this concern.

1.6.1. Introduction to Digital Signature

A digital signature is a way of digitally signing a file or program to ensure it has not been tampered with and that the author is the author claimed. It is analogous to ordinary physical signatures on paper, but implemented using techniques from the field of public key cryptography [14].

There are three common reasons for applying a digital signature to communications:

- Authenticity – allow the recipient of a message to be confident that the sender is who the sender claims to be;
- Integrity – the recipient examines the message to make sure it has not been altered in transit which is called data integrity checking;
- Non-repudiation – the signer cannot later disclaim any knowledge of the message.

1.6.2. Message Digests and Digest Algorithms

Since the public-key cryptography is pretty slow, it is better to encrypt a representative of the data instead of encrypting the entire data. The representative of the data is called message digest in cryptography. The use of a digital signature requires a digest algorithm.

There are many digest algorithms but three important digest algorithms have dominated the market, MD2, MD5 and SHA-1. Ron Rivest created MD2, MD5 and played a role in the design of SHA-1.

MD2 and MD5 are 16-byte digests. Since flaws and collisions have been discovered with MD2, it is not recommended to use MD2 in new applications. MD5 is much faster and much stronger than MD2, and as such it has become the dominant algorithm and still in common use.

SHA-1 contains stronger internals than MD5 and it produces a 20-byte digest which is longer than that of MD5. So it is highly recommended by the cryptographic community. In the DFT system, SHA-1 algorithm is used.

1.6.3. Signature Algorithms

A general digital signature scheme consists of three algorithms:

- A key generation algorithm

- A signing algorithm
- A verification algorithm

RSA, DSA and ECDSA are the three most successful signature algorithms. With RSA, the algorithm encrypts the digest with a private key to produce a digital signature.

With DSA, the signer digests the message with SHA-1. DSA does not encrypt the digest. It has three inputs: The digest which is a number 160 bits long; a random or pseudo-random value, usually called *k*; and the private key. The algorithm then performs some mathematical operations. The output of DSA is two numbers, usually called *r* and *s*. These two numbers are the signature. When verifying the signature, the verifier computes the SHA-1 digest of the message. Using the digest as a number, along with the public key and *s*, the verifier performs some mathematical operations. The result of the computation is a number called *v*. If *v* is the same as *r*, the signature is verified. Figure 1.6.3 depicts the producing and verifying a DSA signature.

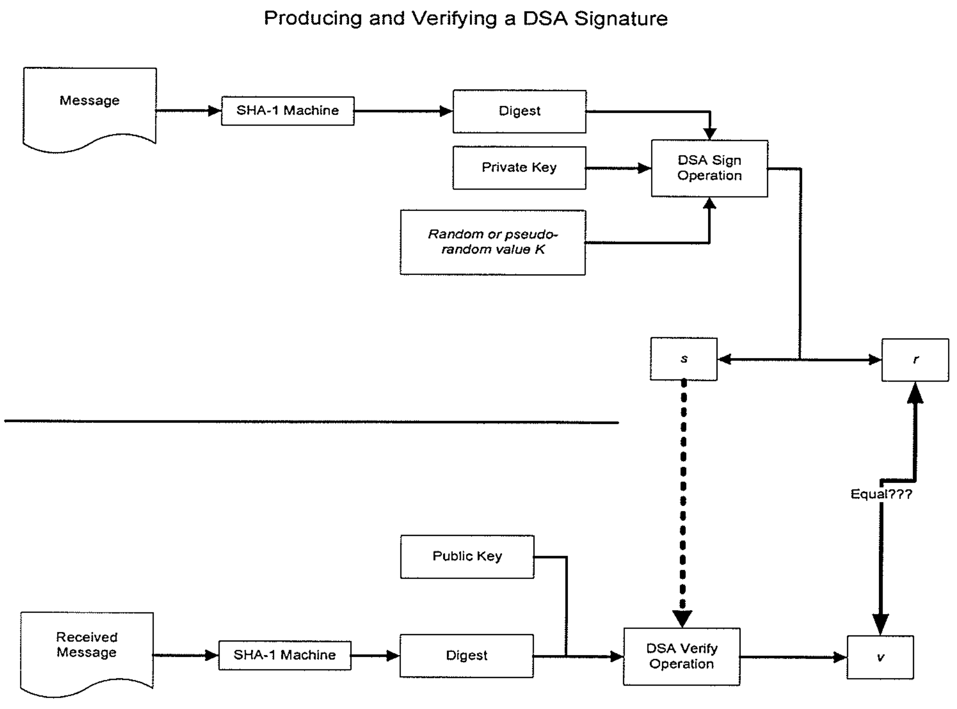


FIGURE 1.6.3 PRODUCING AND VERIFYING A DSA SIGNATURE

ECDSA looks a lot like DSA. It has the same inputs and output numbers as DSA does. If the final v is not equal to r, something went wrong. The difference is the math underlying DCDSA are Elliptic Curve algorithms.

1.6.4. Digital Signatures in Java

The Java class *java.security.Signature* provides signature service and has methods to generate and verify signatures. In the system described here, this class and other classes in the package *java.security* are used to implement the signature generation and verification functionality.

1.7. Summary

This chapter provided an introduction to the knowledge needed to understand this thesis. First a brief introduction to the history of the Internet, the TCP/IP protocols and the birth of Java were given. Then the FTP protocol, its active and passive modes, and its problems in reliability and efficiency were introduced. Lastly the digital signatures and the difference between the mechanism described in this thesis and other download mechanisms is dealt with briefly. Based on this knowledge and the FTP download problems, an improved distributed file download mechanism is proposed in the next section.

Chapter 2: DFT System Objectives and Requirements

Distributed File Transfer (DFT) is designed to improve the traditional FTP and solve the problems that the FTP has. DFT can allocate server resources according to their availability and performance. Therefore, reliability, scalability and efficiency can be improved by making better use of multiple file servers on the Internet. The design objectives and requirements of DFT are briefly described as follows:

1) The system should allow a user to enter a file name. The system should prompt for the file name the user wants to download with a friendly interface.
2) The system should be able to search for the file on the Internet and give the user the option to search a local directory first.
3) The system should display a list of FTP servers that have the required file and let user to select downloading servers from the list.
4) The system should be able to download the file from the selected FTP servers.

5) The system should be able to download the file from each of the selected FTP servers using multiple threads. Each thread downloads one part of the file in parallel. The system should be able to put all downloaded parts of the file together as a complete file.

6) The system should be able to test each FTP servers' speed before downloading.

7) The system should be able to switch to a faster server if a server degrades, or the connection becomes very slow.

8) The system should check whether the files on the selected FTP servers are identical or not before starting the download; if they are not, it will notify the user and download only from the file servers with identical copies.

9) The system should be able to save visited FTP file server information to a local database (info.txt file).

10) The system should allow user to generate a file's digital signature and verify a downloaded file's digital signature.

11) The system should handle exceptions and error properly and log errors into local log file.

Based on these objectives and requirements, the DFT architecture is designed in the next chapter.

Chapter 3: DFT System Architecture

This chapter describes the overall architecture of the DFT system. DFT is a distributed system which includes a DFT client, a Load-Distributing Server (LDS), and multiple outside FTP search engines and FTP servers.

3.1. System Architecture Diagram

Figure 3.1 is the system architecture diagram. It depicts the objects in the system and how they interact with each other.

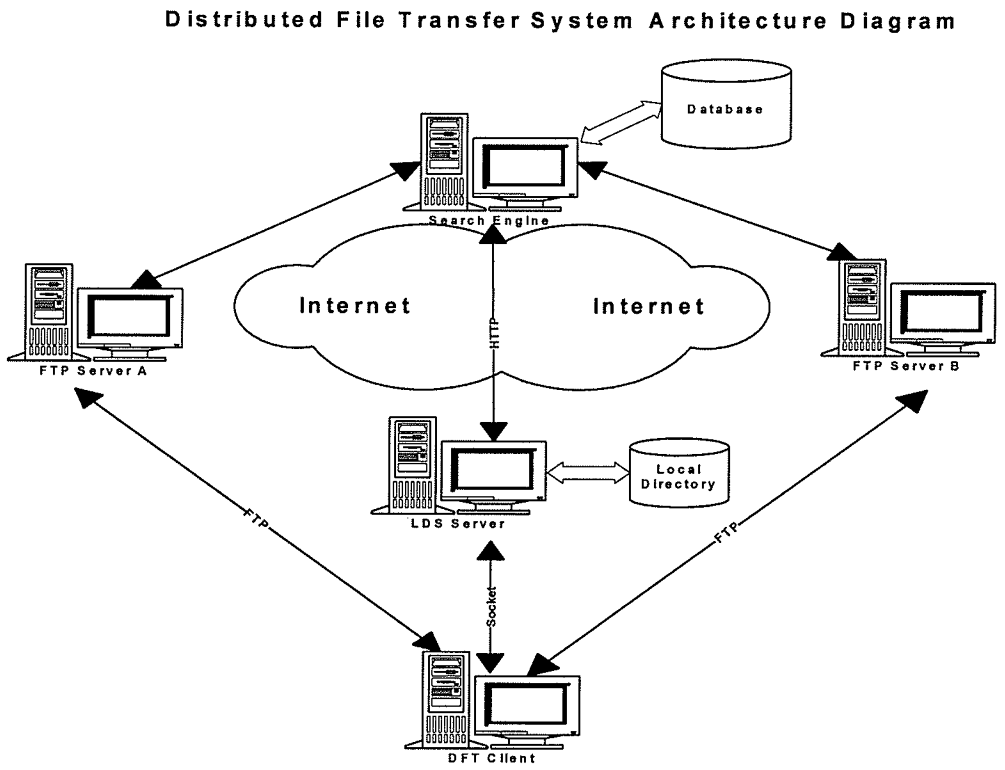


FIGURE 3.1 DFT SYSTEM ARCHITECTURE DIAGRAM

3.2. Distributed Architecture of DFT

DFT is a distributed system. It contains three components: the DFT client, Load-Distributing Server (LDS), and multiple file servers. The DFT client first connects to the LDS server and gets locations of the FTP servers. The DFT client then connects to multiple FTP servers by opening multiple sockets and downloads the file from multiple locations. More specifically, the LDS server receives a request from a DFT client. The LDS server then either retrieves target file location information from its local directory, or goes to an outside FTP search engine to search for target file locations. The user has the option to select from a list of FTP servers for the download. In this scenario, the DFT system depends on the outside FTP search engines for target file location information.

DFT has the following distribution properties:

- Data Distribution–In order to achieve reliability, a file is replicated on many file servers. Each file server supports a segmented transfer of the file. Segmented transfer means a user can request a segment of the file to be transferred 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 of the DFT system can be distributed across the Internet. The DFT client and LDS servers are connected through socket connections. The LDS server connects to FTP search engines using the HTTP protocol. The DFT client connects to FTP servers using the FTP protocol.

- Heterogeneous System–Since the DFT system components can be distributed across the Internet, the DFT system can run on different platforms. The DFT client and the LDS server are implemented in Java, which generates platform independent code.

3.3. Distributed File Transfer (DFT) Client

A DFT client is a software application deployed on a client's computer. The DFT client displays a user interface that allows a user to enter the file name to download and sends the file name to the LDS server to get a list of available FTP servers on Internet. The DFT client then opens multiple TCP connections to multiple FTP servers simultaneously. From each server, the DFT client only downloads a segment of the file. All segments are then assembled into the target file and saved in the local file system. Depending on each FTP server's speed, the DFT client will intelligently allocate the size of each segment. Each segment can be further divided into equal pieces and downloaded simultaneously by multiple sub-threads. The DFT client also monitors the progress of the download and provides the user with feedback.

A DFT client can test the speed of the specified FTP server. The speed is used to determine the target segment size. The faster the server is, the bigger the segment size the DFT client gets. On the other hand, the slower the server is, the smaller the segment size it gets. The DFT client tests the server speed by sending a request to transfer 10240 bytes of the target file from each FTP server. The DFT client calculates the speed of the transfer using following formula:

Speed (byte per second) = 10240 bytes x 1000 / (start time-end time) (ms)

When a FTP server is not available to download or an error occurs, the DFT client will not use that server for download. The load will be distributed to other available servers automatically without the user being aware.

3.4. Two-Layer Multi-Thread Parallel Download

DFT is considered as a two-layer multi-threaded parallel downloading system. The reason it is two-layer is because it has two layers of threads.

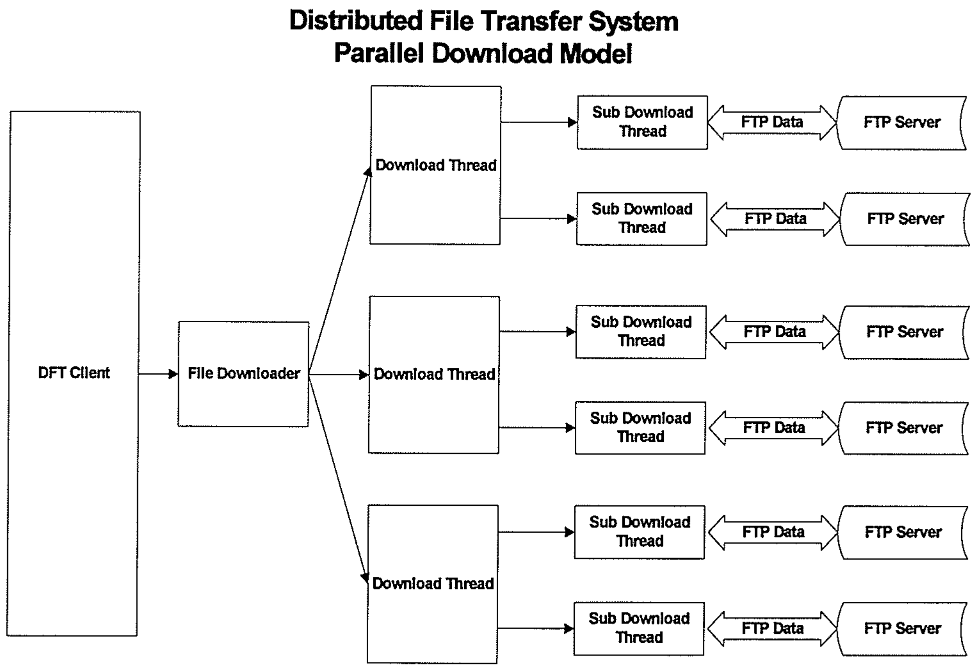


FIGURE 3.4A DFT MULTI-THREAD PARALLEL PROCESSING DIAGRAM

At the beginning of the download, a DFT client sends a query containing the target filename to an LDS server for file server information. The LDS server responses with a list of available file servers that contain the target file. The DFT client then parses the response from the LDS server and determines which file servers to connect to. There is one more step to be completed before the real download starts. The DFT client sends a command to each file server and asks for the first 10240 bytes of the target file. The DFT client records the downloading time for this 10240 bytes and calculates the speed of each file server. The DFT client then allocates a segment size for each file server based on the server's speed.

The DFT client creates one thread for each FTP server that the user has selected to download from. The DFT client uses that thread to download a segment of file from that FTP server. This is first layer of parallelism.

For each downloading thread, the DFT client gives the user the option to set the number of sub-threads. The DFT client then opens a number of sub-threads for that first layer downloading thread. Each sub-thread will download an equal sized sub-segment of the segment. All sub-segments are assembled into the segment. This is the second layer of parallelism. Then all segments are assembled into the whole file. Here, the DFT client uses the checkpoint technique to download a segment of a file.

By using this two-layer multi-threaded downloading, DFT takes full advantage of its parallel processing power. Parallel download enhances the performance and reliability of the download.

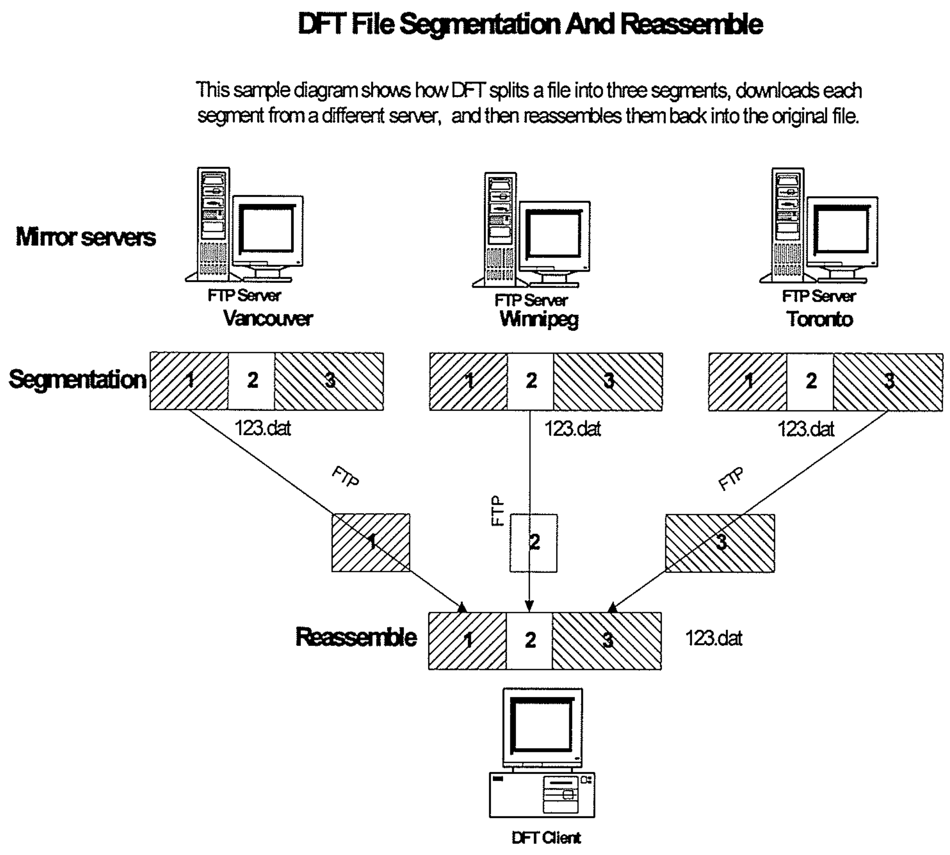

DFT File Segmentation And Reassemble
This sample diagram shows how DFT splits a file into three segments, downloads each segment from a different server, and then reassembles them back into the original file.
Mirror servers
FTP Server
Vancouver
FTP Server
Winnipeg
FTP Server
Toronto
Segmentation
1
2
3
123.dat
FTP
Reassemble
DFT Client

FIGURE 3.4B DFT MULTI-THREAD PARALLEL PROCESSING DIAGRAM

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 may vary over time. As such, the speed calculation is just an estimate. It is very often that one thread finishes faster than another thread.

While downloading is in progress, all data connections have a timeout value. When the server timeout value is reached and no data is transferred, the data connection will terminate its download and report to the control connection. The end user will be notified about the failure.

3.5. Load-Distributing Server (LDS) Server

With FTP, the user has to know where the target file is located. Typically the user can only download a file from one known FTP server. In the DFT system, the user doesn't have to know where the target file is located. They just need to know the file name and the LDS server will find the locations of the file for them.

The LDS server has two major functions. One is looking for the target file in a local directory or across the Internet. The other is to maintain a local directory.

For the former, the DFT client sends a request including the file name to the LDS server through a socket connection. The LDS server reads the download file name from the DFT client. Depending on the user's options, the LDS server will search a local directory or skip the local search. If the LDS server finds the file name in local directory, it will send the file location information (including FTP server address, path, file name, size and time) back to the DFT client. If the LDS server finds more than one entry in the local directory, it sends back all the entries. If the file is not found in the local directory or the user selects skip the local directory to search the Internet directly, the LDS server will open an HTTP connection to one of the registered FTP search engines. The LDS server sends an HTTP request to the FTP search engine, and gets the HTTP response back. The response HTML page is parsed and a list of available FTP server addresses will be sent back to the DFT client through a socket connection. If one FTP search engine failed to find the file, the LDS server will try another search engine. If all search engines fail to find the file, the user will be notified.

The second main function of LDS server is to maintain a local directory that contains information about recent downloaded files. The local directory is in a text file called info.txt. Each entry contains information about the target file. Information includes the file name, FTP server address, path, user name, password, file size and time stamp.

3.6. DFT File Query and Response Model

Figure 3.6 depicts the DFT system file query and response model.

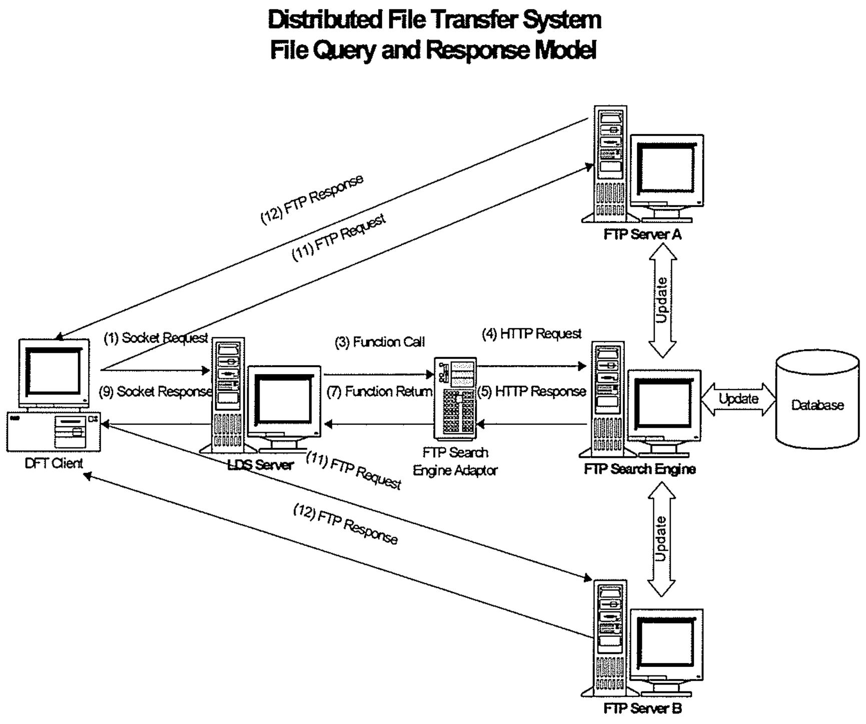


FIGURE 3.6 DFT FILE QUERY AND RESPONSE MODEL DIAGRAM

The above diagram illustrates the steps how DFT client sends out a file query request and finally gets the file.

1) The DFT client sends a socket query request to the LDS server asking for a specific file.
2) The LDS server receives the request.
3) The LDS server creates a FTP search engine adaptor and makes a call to do a search.
4) The FTP search engine adaptor sends a HTTP request to an FTP search engine.
5) The FTP search engine finds the specified file in its database and returns a HTTP response.
6) The FTP search engine adaptor receives and parses the HTTP response.
7) The FTP search engine adaptor returns a list of file servers back to the LDS server.
8) The LDS server gets the returned list of file servers.
9) The LDS server sends the list of file servers back to the DFT client.
10) The DFT client creates multiple threads and starts the download.
11) Each download thread sends an FTP request to one of the FTP file servers.
12) The FTP file server responds with an OK status and starts sending the file.
13) Each download thread reads its segment of file from the FTP server data port.
14) The DFT client reassembles all segments into a complete file.
15) The user can stop and resume the download at any time during the download.
16) The download ends.

The FTP search engines play an important role in the DFT system. Since the user doesn't know where target file is located prior to their download, the DFT system depends on a search engine to find file locations. An FTP search engine generates an HTML page that contains a list of FTP server addresses and sends back an HTML page as a response.

A general list of FTP search engines from Internet may not be working in the future as they are not well-maintained production websites. In this work a few working ones were selected as the search engines. As better ones are found, the list can be updated.

The DFT client gets the list of FTP server addresses and uses them to open FTP sessions. The DFT client opens one session for each server. The DFT client uses standard FTP commands to send requests and receive data.

By using multiple FTP servers for downloading, a kind of redundancy is added to the system. A target file could be available on many mirror sites. Should one or more FTP servers fail to operate, DFT can always go to other FTP servers for downloading the same file.

3.7. DFT Two-Layer Control Connection Model

Figure 3.7 depicts the DFT two-layer control connection model.

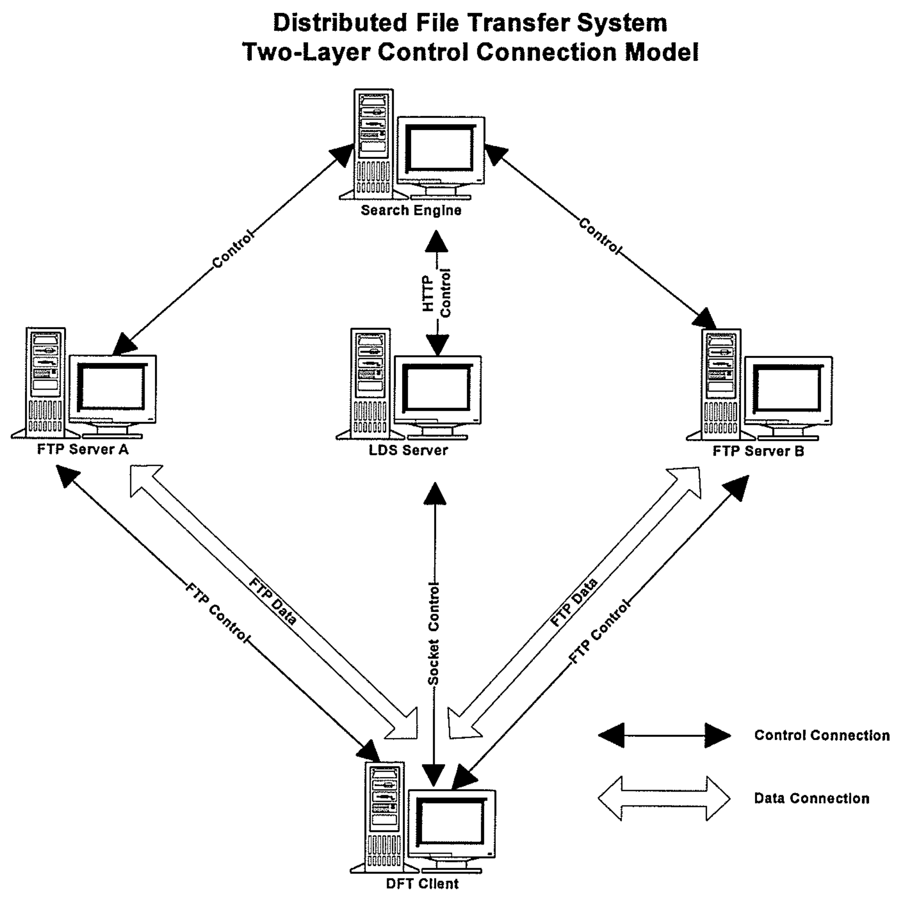


FIGURE 3.7 DFT TWO- LAYER CONTROL CONNECTION MODEL

Traditional FTP uses two TCP connections. One is control connection at port 21 to send FTP control commands and receive responses. Another one is a data connection that transfers a file between client and server. The traditional FTP connection model was illustrated in a previous chapter. Here the DFT connection model is explained.

As the DFT system is a distributed reliable file transfer architecture (i.e. there is more than one file server involved), it is necessary to enhance the control connection part of FTP to work for this distributed architecture.

The control layer of DFT includes control commands from the DFT client to a FTP server as well as control command from the DFT client to the LDS server. Before a DFT client begins downloading a file, it does not know where the file is located. Through a control layer connection, the DFT client sends a query to the LDS server and gets a list of FTP servers and target file locations. Then the DFT client makes control layer connections to the FTP servers and gets their responses. Then the DFT client makes data connections to the FTP servers and starts the file downloading.

When the data transfer is finished, the DFT client sends the download FTP server information to the LDS server through control-layer connections. The LDS server then updates its local directory information.

Generally control-layer connections are established between a DFT client and the LDS server. However, when the target file is not found in the LDS's local database, the LDS will make a connection to one of the search engines and look for the target file in that search engine. This kind of connection is classified as a control-layer connection as well.

Before downloading starts, the DFT client will retrieve a small part of the target file to calculate the speed of the file server. If the server is not available, the DFT client will delete the server from the list and use other file servers. This is done by establishing traditional FTP control and data connections between the LDS server and file server. This is considered another kind of DFT control-layer connection.

If the target file is downloaded from more than one file server, the DFT system has the ability to check if the files on the selected mirror sites are identical by comparing their file sizes and the first 10240 bytes. If they are not, download will be aborted. Alternatively a digital signature could be implemented as discussed. Implementing digital signature is not possible on commercial FTP servers.

The following functions are integrated into the control layer.

1) LDS request/reply
2) Server status check
3) Download process monitoring
4) Server speed calculation
5) Server availability
6) Download task allocation
7) Failure detection
8) Download resume/recovery

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

With this model, multiple data connections can be established between the DFT client and several file servers.

3.8. DFT Data Connection Model

When a DFT client downloads a file from one of the file servers, it establishes a data connection with the file server. The target file is transferred through a data connection. Since a multiple-thread parallel download model is used, the DFT client can establish multiple data connections to each file server. Each data connection is responsible for one segment of the target file.

While the data connections are performing the file download, the control-layer keeps monitoring the status of each data connection. If there is any abnormal behavior, the status will be reported to the end user.

The data layer concentrates on the following functions.

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

3.9. Failure Detection

Because of the distributed architecture of the DFT system, different components are connected to each other using HTTP, FTP, or pure socket connections. It is important to design a mechanism to detect network failures, and application failures.

Failure detection can take place on both the control layer and data layer. When failure happens on the control layer, the control layer tries to recover from it immediately or reports to a log file or the end user. When failure happens on the data layer, the data layer will stop the download process and notify the control layer.

When something is not working well, the DFT system normally generates one of two kinds of failures: network failures and application failures.

3.9.1. Two kinds of failures

This section discusses network failures, application failures and responses to those events.

3.9.1.1. Network failures

Network failures are caused by network hardware failure, data connection timeout, or control connection timeout. When this kind of failure happens, the client and server are disconnected. The server cannot receive requests from the client and in the meantime, the client cannot receive response from the server.

3.9.1.2. Application failures

Application failures are less severe than network failures. When application failures occur, the client and server are still connected. It's just that the response from server is not what client is expecting. The client and server can still communicate to each other and tell each other what is wrong.

3.9.2. Two kinds of failure detection methods

Network failures and application failures are handled differently.

3.9.2.1. Timeout detection

When network failure occurs, there will be no messages transferred between the client and server. TCP timeouts will try to recover the connection by requesting a retransmission. However, this doesn't work if the network is down. The client will be waiting for the server response indefinitely. To solve this problem, the DFT client sets a timeout value on each connection. DFT also sets a timer to periodically monitor if the timeout value is reached. When timeout is reached, the DFT client will stop the download process and notify end user of the timeout.

There are two types of timeouts, control layer timeouts (command-response timeout) and data layer timeouts. Command-response timeouts occur on control connections. After a DFT client sends out a FTP command, it starts a timer. If a timeout occurs before it receives the response, the DFT client stops this control connection. Data layer timeouts occur on data connections. A timeout is set when a data socket connection is opened. If the elapse time between two packets is longer than the timeout, the client assumes the data connection is broken or the network is congested.

3.9.2.2. Application failure code detection

Application failure code is used to detect application errors. A common scenario is as follows. When a DFT client makes a connection or sends a FTP command to a file server, it expects to receive a response in the form of a string. The string has a certain format that follows the FTP protocol standard. The string will begin with three-digit number followed by a space and an error message. A typical response is "226 Transfer complete". The three-digit number can be parsed and thus it is known what error code it is. A code larger than 400 generally indicates an incomplete service. A list of FTP error codes is provided in Appendix C.

Through application failure code detection, all application failures can be detected immediately once they happen. Such failures include:

- Server not available
- File not found
- Too many users
- Login failure
- Connection closed
- Cannot open a connection
- Broken Pipe

3.10. Failure Recovery

When the control layer detects a failure or receives a 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

If a hardware failure occurs, the DFT client itself will be terminated. In this case, the DFT client has to be run again to resume the terminated download process. To recover from such a failure, the checkpoint of the last download and connection status information must be recorded. The recover process will read these status records and resume all previous connections. To implement such a recovery mechanism, the DFT client periodically records the status of all connections on the user's hard drive.

3.11. Checkpoint Resume

The DFT client uses the Checkpoint Resume service to implement the multi-threaded parallel download function. The DFT client cuts the file into multiple segments based on the FTP servers' speed and each server is assigned to a segment. The DFT client creates multiple threads to download the segment of a file from each server. The segment is divided into sub-segments based on the thread number. Each thread opens one session and downloads a part of the server's segment of the file. The DFT client remembers the checkpoint of each segment for each thread. When resuming the download, it restarts according to the checkpoint of each thread.

3.12. Local Directory

DFT maintains searched results into a local directory (info.txt file). Info.txt is a text file following a certain format. Each line of the file represents one target file's information. Each line contains multiple fields separated by a colon. The format of each line is:

```
[File Name]:[IP Address]:[Full Path]:[User Name]:[Password]:[File Size]:[Time]:
```

Sample line:

```
icqpro2003b.exe:192.168.123.1:/yuhong:anonymous:a@a.com:2000:12312003:
```

Here each line contains enough information for DFT to download the target file from a specified file server. The purpose of Local Directory is to store searched results locally as a kind of cache so that DFT doesn't have to go to a search engine every time and thus improves the overall download time. The user has the option not to save searched results into the Local Directory.

3.13. Digital Signature

The DFT system has the ability to download files as well as their digital signature and verify their digital signatures. The DFT system has the ability to generate a file's digital signature and upload it to the file server. An assumption is that the file server is responsible for providing a digital signature for the target file. Because of circumstance limitations, an environment is imitated for testing. An IIS FTP server is set up and the test files and their digital signature files are uploaded to the server. The DFT client is run and downloads the target files as well as its digital signature. Finally, the downloaded file is verified against its digital signature. Thus it can be made certain whether the target file is the one expected.

The operation of the IIS FTP server is maintained throughout the development cycle.

3.14. Error Handling

Each module in a DFT application handles errors and exception in a consistent way. All errors and exceptions are written to a local log file called LDS_Logfile.yyyy.mm.dd.

Each line in log file contains following fields.

```
[Time Stamp]|[level]|[Module Name]|[Function Name]|[Message]
```

Sample log file:

```
LDS_Logfile.2004.07.10

2004-07-10-03:57:03-PDT|level=info|module=LDSConnection|method=getFileInfo| Target
file is sent to lds

2004-07-10-03:57:03-PDT|level=info|module=LDSServerThread|method=run|
TargetFileName is icqpro2003a.exe

2004-07-10-03:57:04-PDT|level=error|module=FreeWareWeb|method=doSearch| found target
file on ftp server -
ftp.carrier.kiev.ua/pub/windows/icq/ICQ/ICQ_Win95_98_NT4/ICQ2000a/icqpro2003a.exe
```

3.15. Summary

This chapter has presented the overall architecture of DFT that was developed for this thesis. Three major parts of DFT system and their relations were introduced. A two-layer control model, which is different from a traditional FTP control model, is described. In addition, parallel download, failure detection and recovery is described as well. A brief overview of digital signature implementation is also given. Based on this architecture, the DFT system implementation is described in the next chapter.

Chapter 4: Implementation

This chapter describes implementation details of all DFT components.

4.1. Implementation Tools

Java was chosen as the programming language to implement the DFT client and the LDS server. In Chapter 1, the birth of Java was discussed. Here further exploration in depth of the language itself will be given. The Java IDE used in the implementation of the DFT system is also mentioned.

4.1.1. Java Programming Language

Java provides developers with many features. They include:

- Object-orientation. Java is an Object Oriented programming language. All executable code must be contained within a class. Java incorporates such object-oriented concepts as inheritance, encapsulation, and polymorphism.
- Portability. Java was designed and developed to produce code that would run on variety of CPU's and under different operating environments without alteration. Java Programs are platform-independent.
- Multi-threaded. Java supports multi-threaded programming. This is important when designing interactive, networked programs or when running multiple applets in a web page.

- Automatic garbage collection. In C++, once a programmer has created and used an object he needs to destroy it to avoid using unnecessary memory. This is not the case with Java. Java has a build-in garbage collector – the Java virtual machine runs a garbage collection algorithm in the background. The programmers don't have to write destructors which may lead to logical errors in the cleanup code.
- Secure. Java is intended to be secure. Java enables the construction of virus-free, tamper-free programs. A Java program cannot corrupt memory outside of its process space. Java applets cannot access the disks of other computers.
- Network and "Internet" aware. One of the reasons for the popularity of Java is that Java is the first programming language to exploit the networked programming environment. It provides extensive classes that making network programming easy.
- Simplicity and ease-of-use. Java borrowed a good deal of syntax from C and C++. The developers of Java wanted to produce a simple language, so many of the less useful, on more esoteric features of C and C++ were removed.

Java comes with extensive built-in libraries which are called packages. The packages that come with the Java Development Kit (JDK) contain many hundreds of built-in classes with many thousands of methods. These classes and methods contain commonly used functionality, meaning that a good deal of the programming work has already been done. It remains for the programmers to integrate the built-in classes for their applications.

Java is uniquely suited for network programming and distributed computing. Since connecting machines was one of the main purposes of Java, it was designed and created with extensive networking features. These features make it much easier to access the Internet than any other language. Java's java.net package provides cross-platform abstractions for simple networking operations, including connecting and retrieving files by using common web protocols and creating basic Unix-like sockets. Used in conjunction with its elegant stream-based I/O classes and its easy-to-use multithreading capability, reading and writing files over the network becomes almost as easy as reading or writing files on disk. Network programming has turned from a difficult, highly fiddly black art into a more straightforward process.

4.1.2. Java Foundation Classes (JFC)

The Java Foundation Classes (JFC) are a collection of standard Java APIs for client-side graphics, graphical user interfaces (GUIs), and related programming tasks. It is a part of Java 2 Platform, Standard Edition (J2SE).

The JFC covers the Swing component classes such as those defining buttons and menus, the classes for 2D drawing from the *java.awt.geom* package, and classes that support drag-and-drop capability in the *java.awt.dnd* package. It also includes an API defined in the *javax.accessiblitlity* package that allows applications to be implemented that provide for users with disabilities.

Swing Component APIs extend the AWT to provide a rich, extensible GUI component library with a pluggable look and feel. The pluggable look and feel lets programmers design a single set of GUI components that can automatically have a similar look and feel of any OS platform. Swing components include both 100% Pure Java versions of the existing AWT component set (Button, Scrollbar, Label, etc.), plus a rich set of higher-level components (such as tree view, list box, and tabbed panes).

Swing Component APIs and the other APIs in JFC are used together to enable programmers to build fully functional GUI client applications that run and integrate on any client machine that supports the J2SE platform, including Microsoft Windows, Solaris, Linux, and Mac OSX [15]. The GUIs in this DFT client system was implemented using Swing components. It is thus platform independent and runs on different operating systems that support J2SE.

4.1.3. JCreator Integrated Development Environment

The IDE (Integrated Development Environment) provides great convenience for programmers in their code development. In this project, JCreator was chosen as the Java IDE [16]. JCreator is a powerful IDE for Java development. It is written entirely in C++, which makes it very fast and efficient compared to the Java based editors or IDEs. JCreator provides users with a wide range of functionality such as: Project management, project templates, code-completion, debugger interface, editor with syntax highlighting, wizards and a fully customizable user interface.

With JCreator users can directly compile or run the Java program without activating the main document first. JCreator will automatically find the file with the main method or the html file holding the java applet, and then it will start the appropriate tools.

JCreator has following benefits compared with other Java IDEs:

- Managing projects with an easy to use interface
- Defining customized color schemes for unlimited ways to organize code
- Wrapping around existing projects and allowing user to use different JDK profiles
- Facilitating writing code quickly with project templates.
- Viewing projects with the class browser
- Debugging with an easy, intuitive interface instead of DOS prompts
- Easy configuration of Java tools
- Lower system requirements, but with faster speed

4.2. DFT Package Implementation

In this section, the DFT client and LDS server package implementations are described in detail.

4.2.1. DFT Package Directory Structure

The DFT project is developed under package named *dft*.

The DFT project is split into six sub-packages.

• dft.gui	- contains modules to build GUI interface
• dft.client	- contains modules to perform client side tasks
• dft.server	- contains modules to perform server side tasks
• dft.util	- contains common utility modules to be shared by other modules in DFT
• dft.gensig	- contains modules to generate digital signature
• dft.versig	- contains modules to verify digital signature

4.2.2. DFT Packages

This section introduces each package in detail.

4.2.2.1. Package dft.gui

This package contains modules to create client side GUI interface including Frames, Panels, Tables, Dialog boxes. User interacts with these GUI interfaces when performing a file download.

```
SelectPanel.java
DFTFrame.java
SelectionTableModel.java
SelectFrame.java
DFTFrame_AboutBox.java
DFTClient.java
```

4.2.2.2. Package dft.client

This package contains client side modules to perform download tasks. These modules make connections to the LDS server, or FTP servers, create download threads, send control commands, read server response, and get data from file servers.

```
SubDownloadThread.java
CommandException.java
FileDownloader.java
ControlConnection.java
DataConnection.java
DownloadThread.java
FileInfo.java
LDSConnection.java
LogRecord.java
```

4.2.2.3. Package dft.server

This package contains the server side modules to perform the target file query and search functions. These modules listen for the query requests from the clients and make connections to the search engines to find out the target file locations, or read file information from a local directory.

```
FileWatcher.java
FTPSearchEngine.java
FileSearching.java
FreeWareWeb.java
OreonRu.java
LDSServerThread.java
Veoda.java
Elmundo.java
LDSServer.java
```

4.2.2.4. Package dft.util

This package includes some common utility modules. LogManager defines common logging functions. LDSConstants defines public constants shared among all modules.

```
LogManager.java
LDSConstants.java
```

4.2.2.5. Package dft.gensig

This package defines a module to generate digital signatures.

```
GenSig.java
```

4.2.2.6. Package dft.versig

This package defines a module to verify digital signatures.

```
SigChecker.java
```

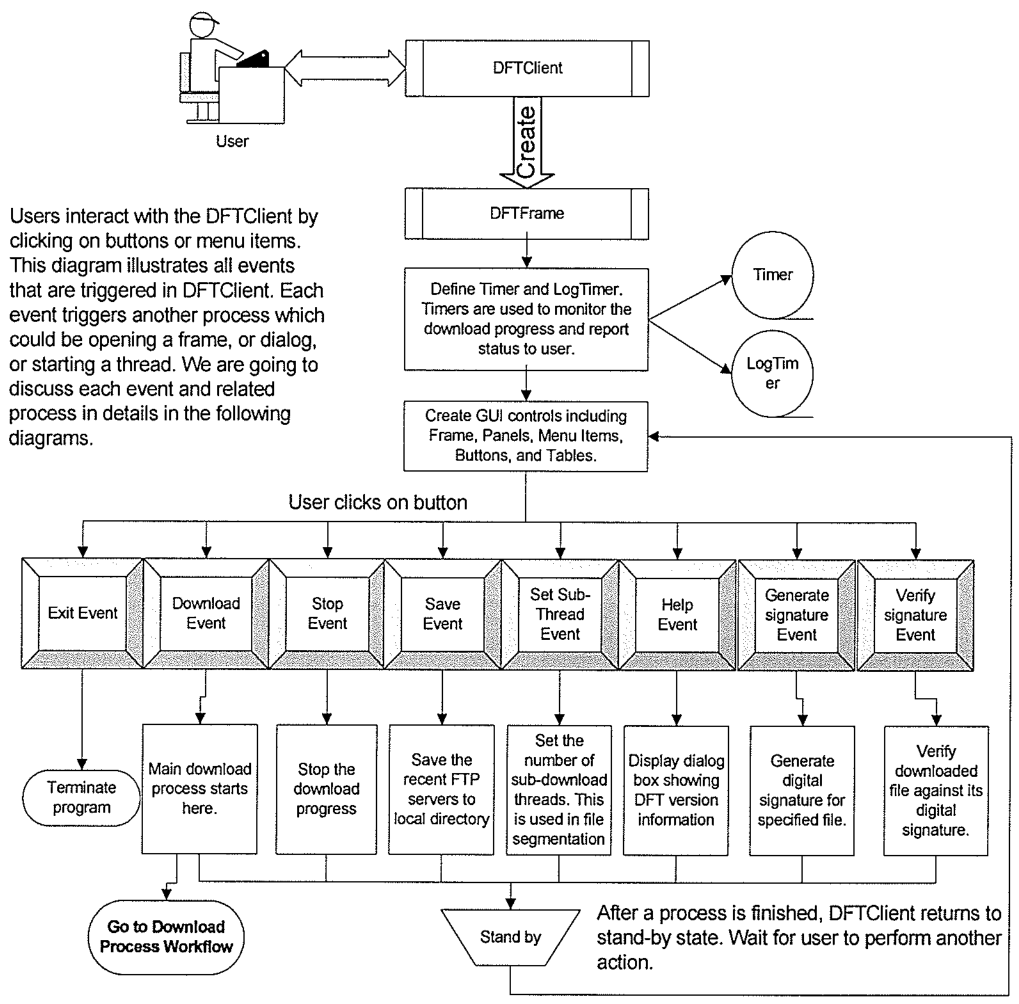
4.3. DFT Modules Implementation

This section talks about major DFT modules and their workflows in detail.

4.3.1. DFT Client Side Implementation

4.3.1.1. DFT Client Workflow

DFT Client Workflow (Main Diagram)


User
DFTClient
Create
DFTFrame
Define Timer and LogTimer. Timers are used to monitor the download progress and report status to user.
Timer
LogTimer
Create GUI controls including Frame, Panels, Menu Items, Buttons, and Tables.
User clicks on button
Exit Event
Download Event
Stop Event
Save Event
Set Sub-Thread Event
Help Event
Generate signature Event
Verify signature Event
Terminate program
Main download process starts here.
Stop the download progress
Save the recent FTP servers to local directory
Set the number of sub-download threads. This is used in file segmentation
Display dialog box showing DFT version information
Generate digital signature for specified file.
Verify downloaded file against its digital signature.
Go to Download Process Workflow
Stand by

Users interact with the DFTClient by clicking on buttons or menu items. This diagram illustrates all events that are triggered in DFTClient. Each event triggers another process which could be opening a frame, or dialog, or starting a thread. We are going to discuss each event and related process in details in the following diagrams.

After a process is finished, DFTClient returns to stand-by state. Wait for user to perform another action.

FIGURE 4.3.1.1 DFT CLIENT SIDE WORKFLOW

4.3.1.2. Download Process Implementation

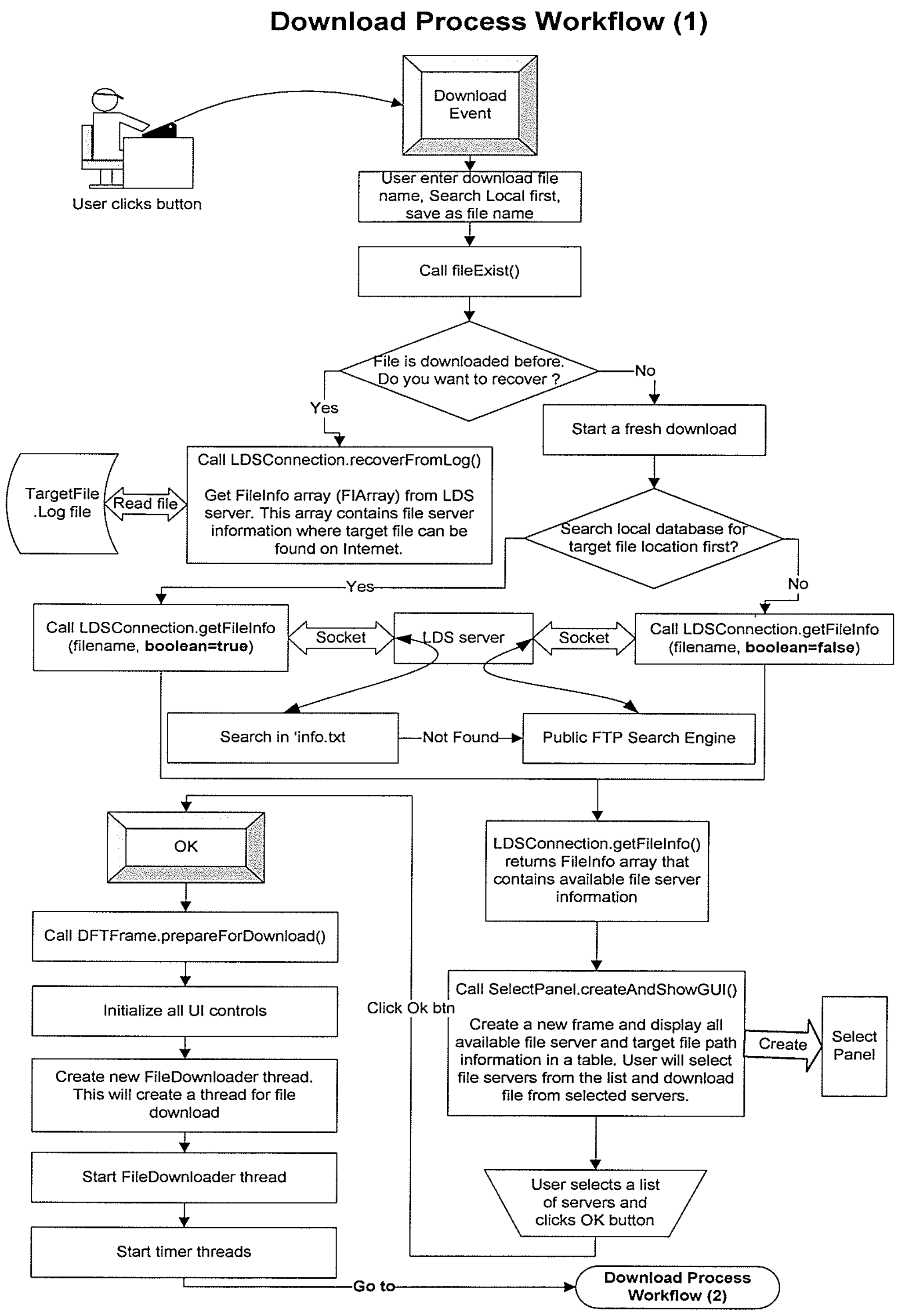

Download Process Workflow (1)
User clicks button
Download Event
User enter download file name, Search Local first, save as file name
Call fileExist()
File is downloaded before. Do you want to recover ?
Yes
No
Start a fresh download
TargetFile .Log file
Read file
Call LDSConnection.recoverFromLog()
Get FileInfo array (FIArray) from LDS server. This array contains file server information where target file can be found on Internet.
Search local database for target file location first?
Yes
No
Call LDSConnection.getFileInfo (filename, boolean=true)
Socket
LDS server
Socket
Call LDSConnection.getFileInfo (filename, boolean=false)
Search in 'info.txt
Not Found
Public FTP Search Engine
OK
LDSConnection.getFileInfo() returns FileInfo array that contains available file server information
Call DFTFrame.prepareForDownload()
Initialize all UI controls
Create new FileDownloader thread. This will create a thread for file download
Start FileDownloader thread
Start timer threads
Click Ok btn
Call SelectPanel.createAndShowGUI()
Create a new frame and display all available file server and target file path information in a table. User will select file servers from the list and download file from selected servers.
Create
Select Panel
User selects a list of servers and clicks OK button
Go to
Download Process Workflow (2)

FIGURE 4.3.1.2A DOWNLOAD PROCESS WORKFLOW (1)

Download Process Workflow (2)

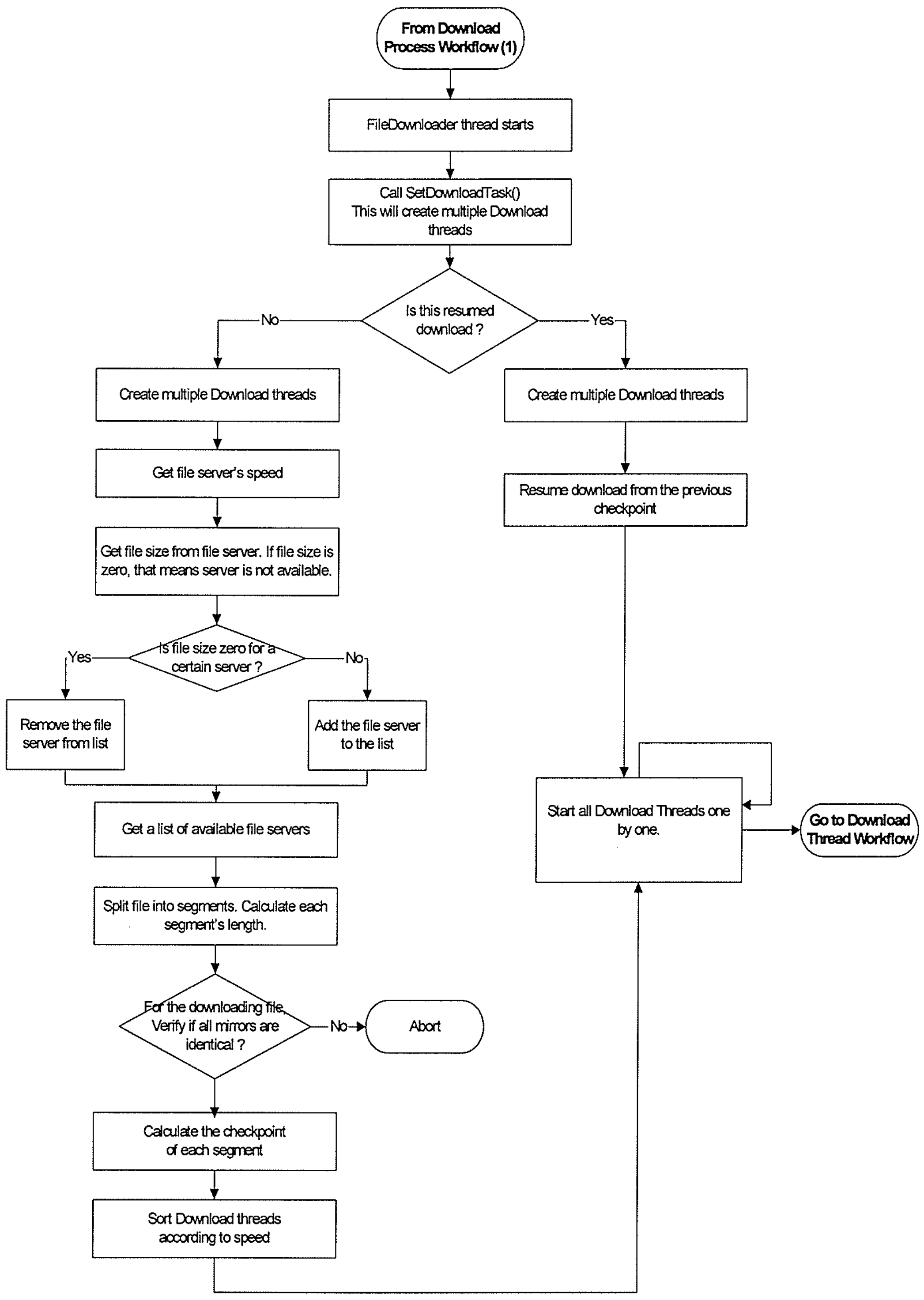

From Download Process Workflow (1)
FileDownloader thread starts
Call SetDownloadTask() This will create multiple Download threads
Is this resumed download ?
No
Yes
Create multiple Download threads
Create multiple Download threads
Get file server's speed
Resume download from the previous checkpoint
Get file size from file server. If file size is zero, that means server is not available.
Is file size zero for a certain server ?
Yes
No
Remove the file server from list
Add the file server to the list
Start all Download Threads one by one.
Go to Download Thread Workflow
Get a list of available file servers
Split file into segments. Calculate each segment's length.
For the downloading file, Verify if all mirrors are identical ?
No
Abort
Calculate the checkpoint of each segment
Sort Download threads according to speed

FIGURE 4.3.1.2B DOWNLOAD PROCESS WORKFLOW (2)

4.3.1.3. Download Thread Implementation

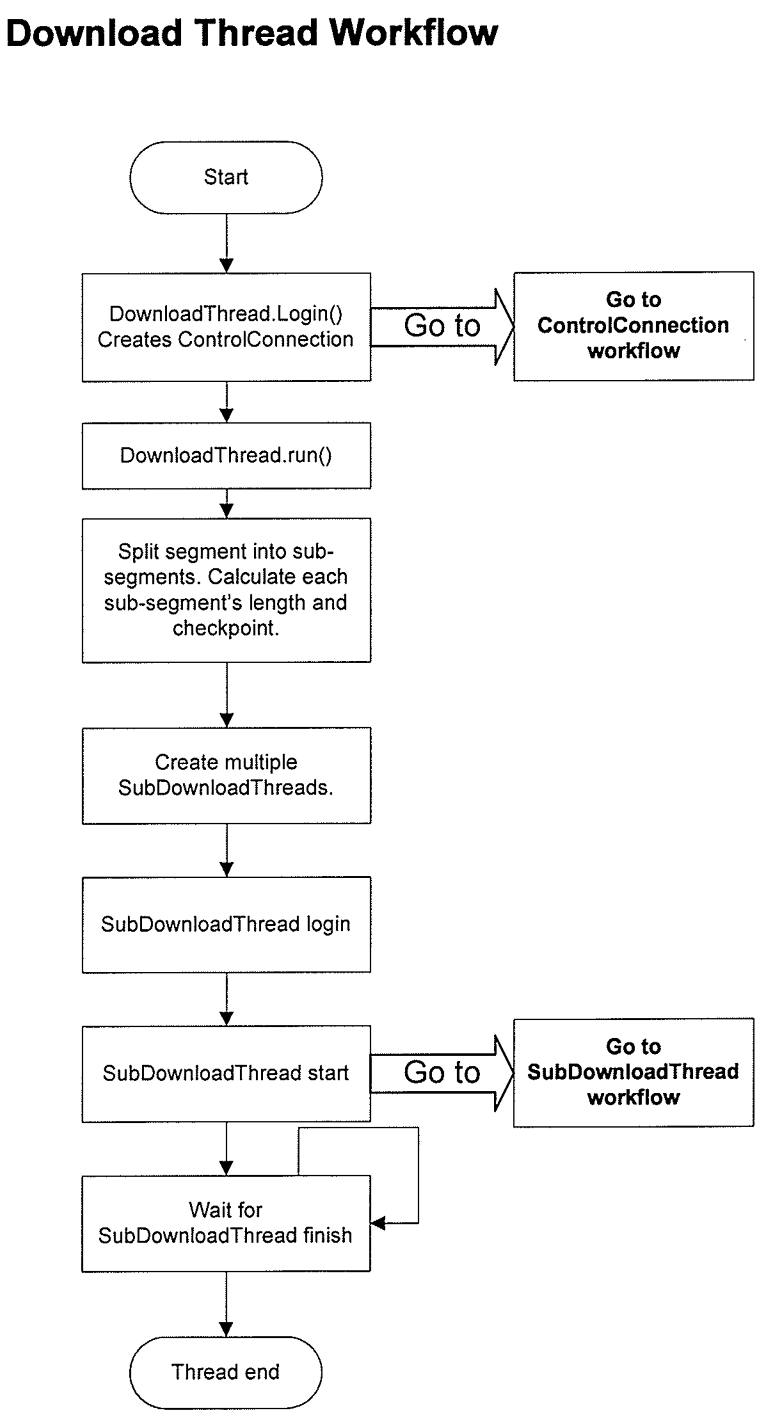

Download Thread Workflow
Start
DownloadThread.Login() Creates ControlConnection
Go to
Go to ControlConnection workflow
DownloadThread.run()
Split segment into sub-segments. Calculate each sub-segment's length and checkpoint.
Create multiple SubDownloadThreads.
SubDownloadThread login
SubDownloadThread start
Go to
Go to SubDownloadThread workflow
Wait for SubDownloadThread finish
Thread end

FIGURE 4.3.1.3 DOWNLOAD THREAD WORKFLOW

4.3.1.4. Sub-Download Thread Implementation

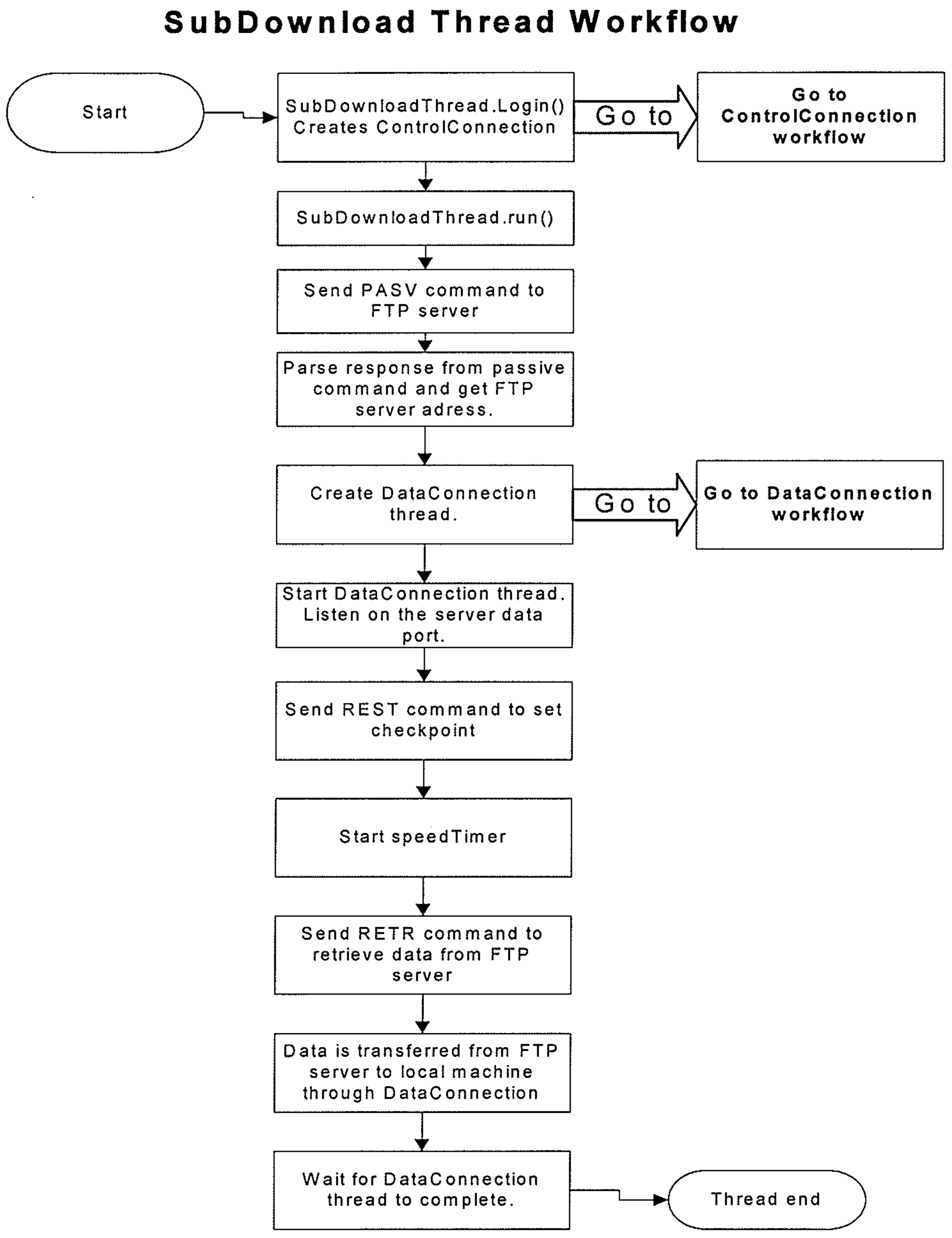

SubDownload Thread Workflow
Start
SubDownloadThread.Login() Creates ControlConnection
Go to
Go to ControlConnection workflow
SubDownloadThread.run()
Send PASV command to FTP server
Parse response from passive command and get FTP server adress.
Create DataConnection thread.
Go to
Go to DataConnection workflow
Start DataConnection thread. Listen on the server data port.
Send REST command to set checkpoint
Start speedTimer
Send RETR command to retrieve data from FTP server
Data is transferred from FTP server to local machine through DataConnection
Wait for DataConnection thread to complete.
Thread end

FIGURE 4.3.1.4 SUB-DOWNLOAD THREAD WORKFLOW

4.3.1.5. DataConnection Implementation

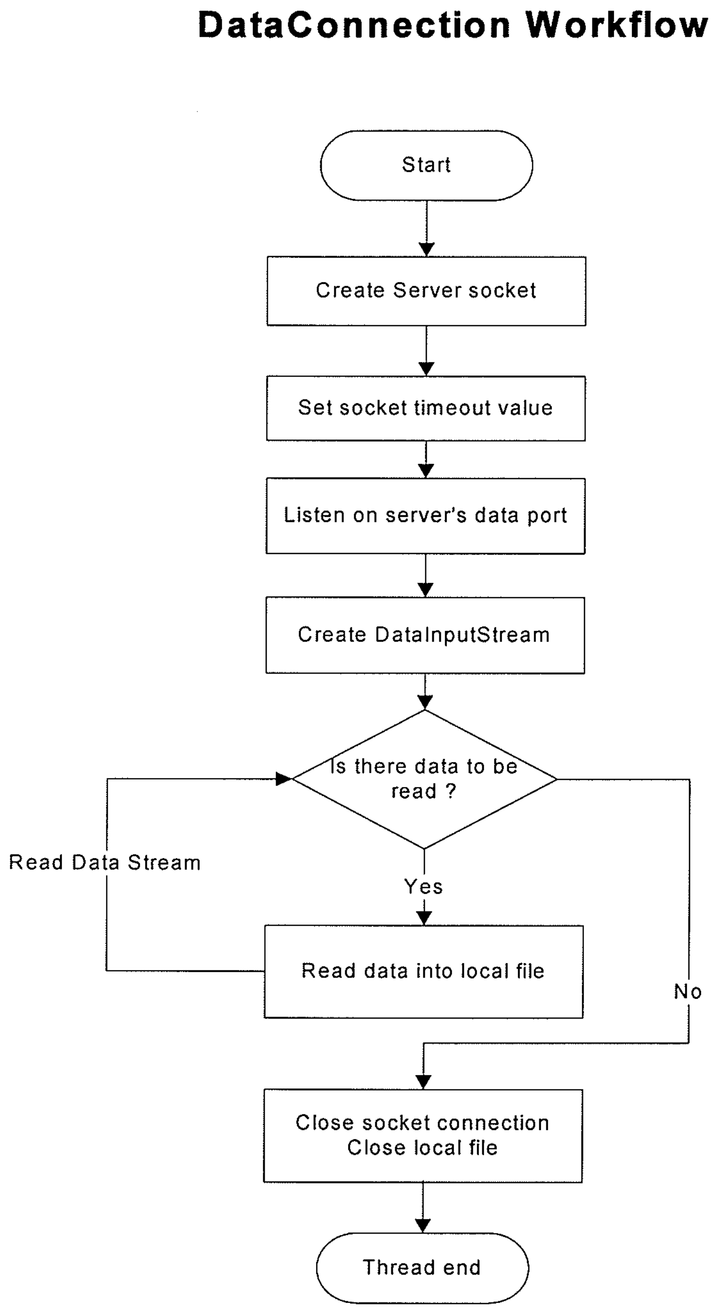


FIGURE 4.3.1.5 DATACONNECTION WORKFLOW

4.3.1.6. ControlConnection Implementation

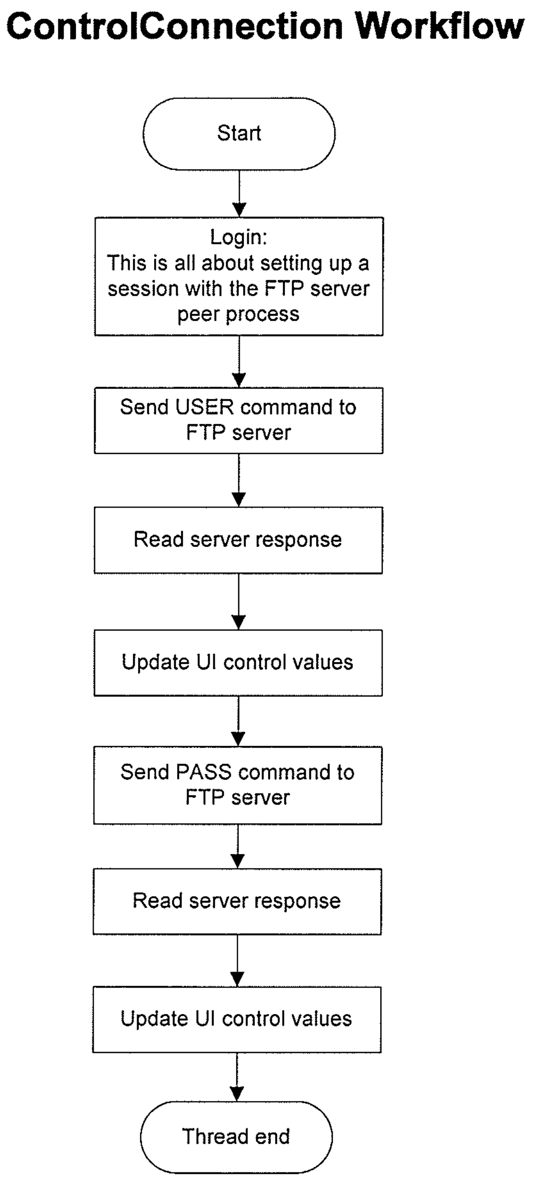

ControlConnection Workflow
Start
Login:
This is all about setting up a session with the FTP server peer process
Send USER command to FTP server
Read server response
Update UI control values
Send PASS command to FTP server
Read server response
Update UI control values
Thread end

FIGURE 4.3.1.6 CONTROLCONNECTION WORKFLOW

4.3.1.7. Timer Implementation

Timers play an important role in the DFT system. A Timer periodically checks the download status and if it sees that the download is complete, the timer will stop itself, close speedLog and display status information to the end user whether it is a success or failure.

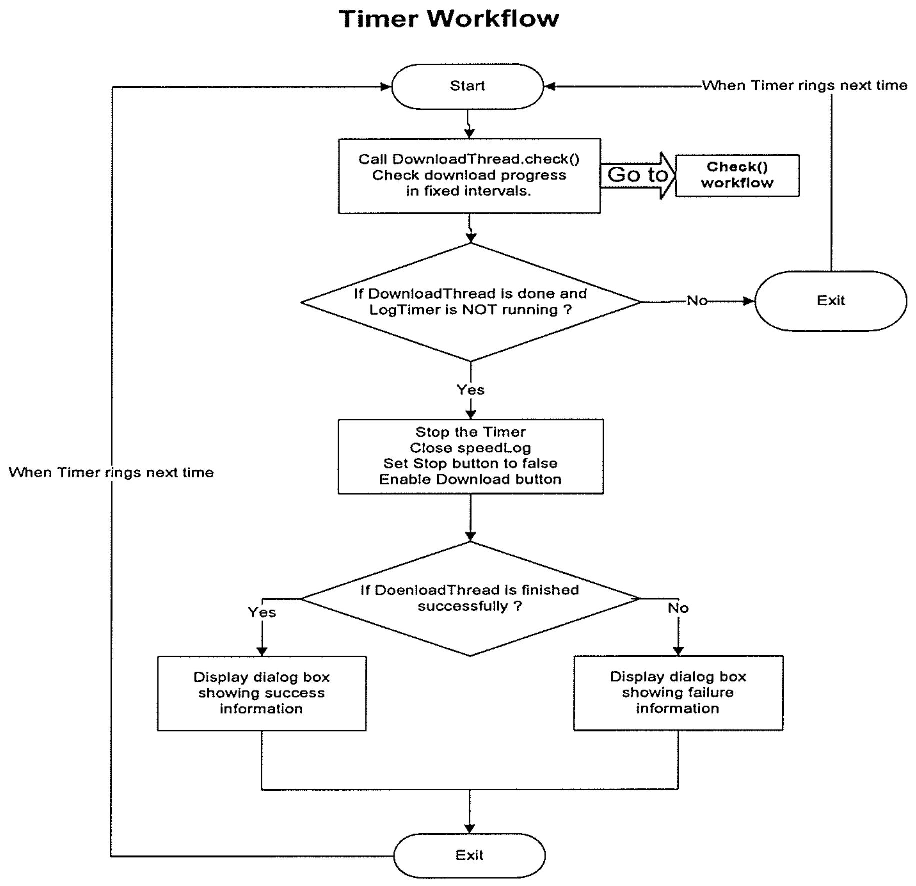

Timer Workflow
Start
When Timer rings next time
Call DownloadThread.check() Check download progress in fixed intervals.
Go to
Check() workflow
If DownloadThread is done and LogTimer is NOT running ?
No
Exit
Yes
Stop the Timer Close speedLog Set Stop button to false Enable Download button
When Timer rings next time
If DoenloadThread is finished successfully ?
Yes
No
Display dialog box showing success information
Display dialog box showing failure information
Exit

FIGURE 4.3.1.7 TIMER WORKFLOW

4.3.1.8. check() Implementation

The method check() checks the status of all download threads and updates the recover log. Each time it is called, the recover log is cleared first and the current download status is written into the recover log so that recover log only contains the latest status. It then checks whether there is any stalled threads. If there is, it will find the fastest server that is being downloaded from and switch the stalled server's task to it.

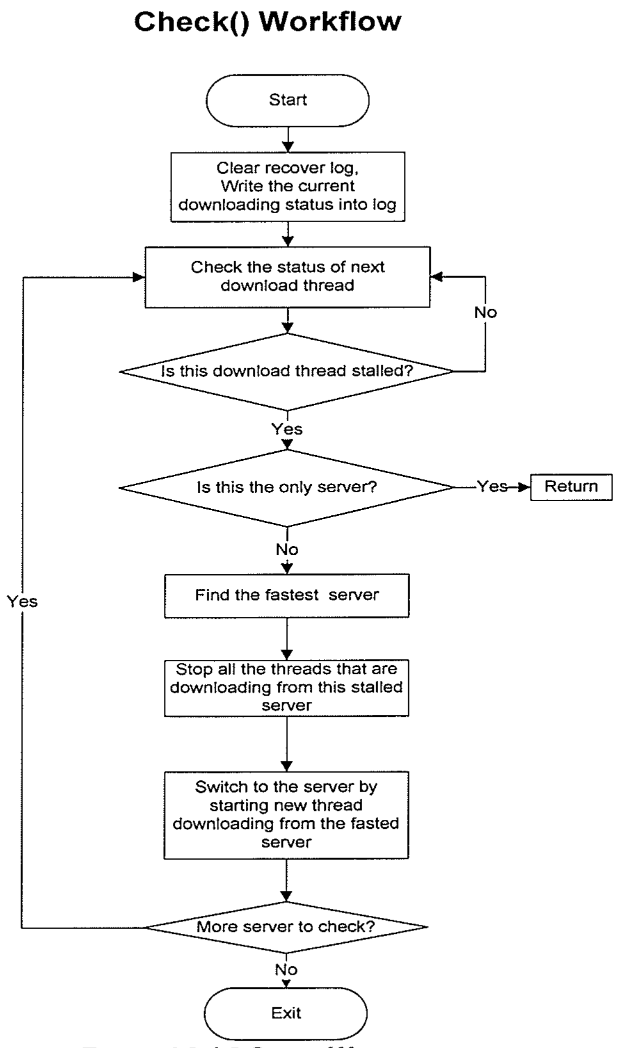

Check() Workflow
Start
Clear recover log, Write the current downloading status into log
Check the status of next download thread
No
Is this download thread stalled?
Yes
Is this the only server?
Yes
Return
No
Find the fastest server
Yes
Stop all the threads that are downloading from this stalled server
Switch to the server by starting new thread downloading from the fasted server
More server to check?
No
Exit

FIGURE 4.3.1.8 CHECK WORKFLOW

4.3.1.9. LogTimer Implementation

LogTimer plays an important role in the DFT system. LogTimer periodically updates the DFT Client UI display while the download is in progress. The user can see how the progress bar is moving as the download file size is increasing. The FTP command and response are shown in the display window. All of these are triggered by LogTimer.

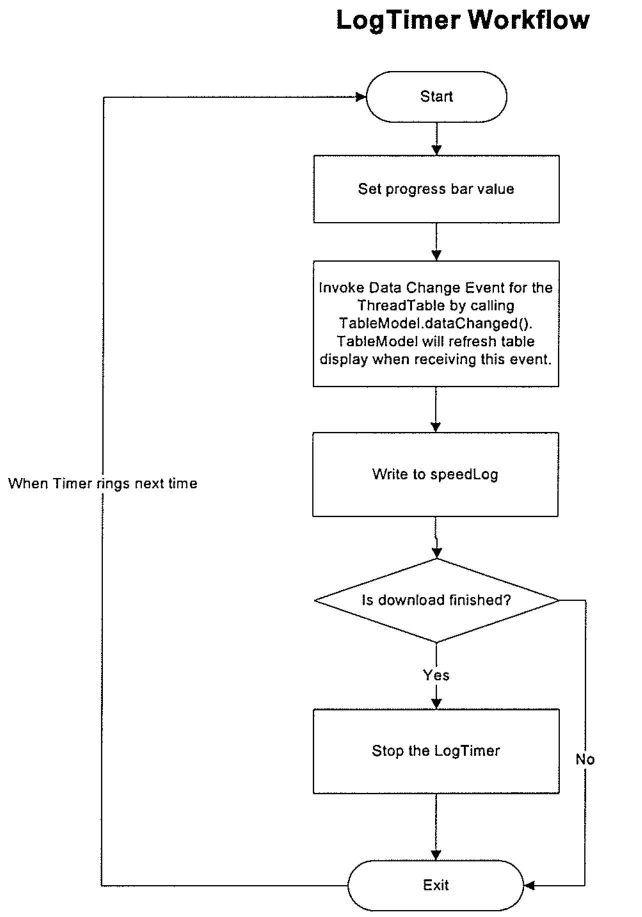

LogTimer Workflow
Start
Set progress bar value
Invoke Data Change Event for the ThreadTable by calling TableModel.dataChanged(). TableModel will refresh table display when receiving this event.
Write to speedLog
Is download finished?
Yes
Stop the LogTimer
No
Exit
When Timer rings next time

FIGURE 4.3.1.9 LOGTIMER WORKFLOW

4.3.1.10. Sub-Download SpeedTimer Thread Implementation

A SpeedTimer started in sub-download thread plays an important role in the DFT System. A SpeedTimer periodically checks the download speed of each thread. When the speed drops to less than half of the original test speed, it sets a flag in the thread and stops the control and data connection of this thread. Then the timer thread will switch the task that the threads are downloading from this server to a fast server.

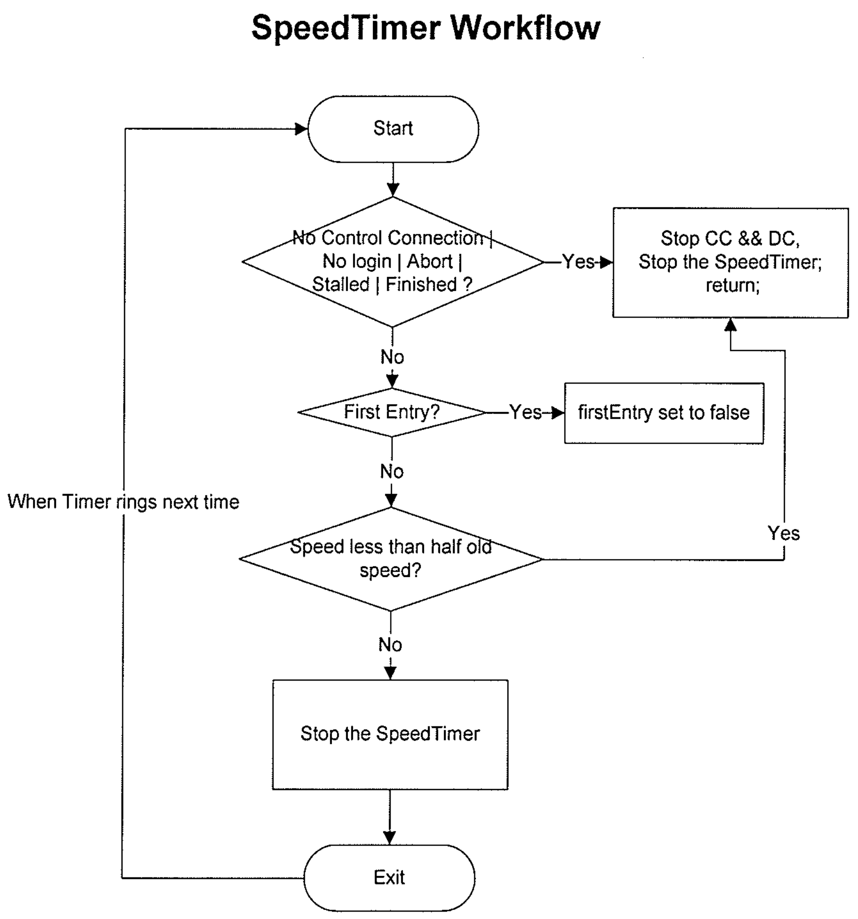

SpeedTimer Workflow
Start
No Control Connection | No login | Abort | Stalled | Finished ?
Yes
Stop CC && DC, Stop the SpeedTimer; return;
No
First Entry?
Yes
firstEntry set to false
No
When Timer rings next time
Speed less than half old speed?
Yes
No
Stop the SpeedTimer
Exit

Figure 4.3.1.10 Sub-Download SpeedTimer Thread Workflow

4.3.2. DFT Server Side Implementation

This section describes the LDS server side implementation in detail.

4.3.2.1. LDS Server Implementation

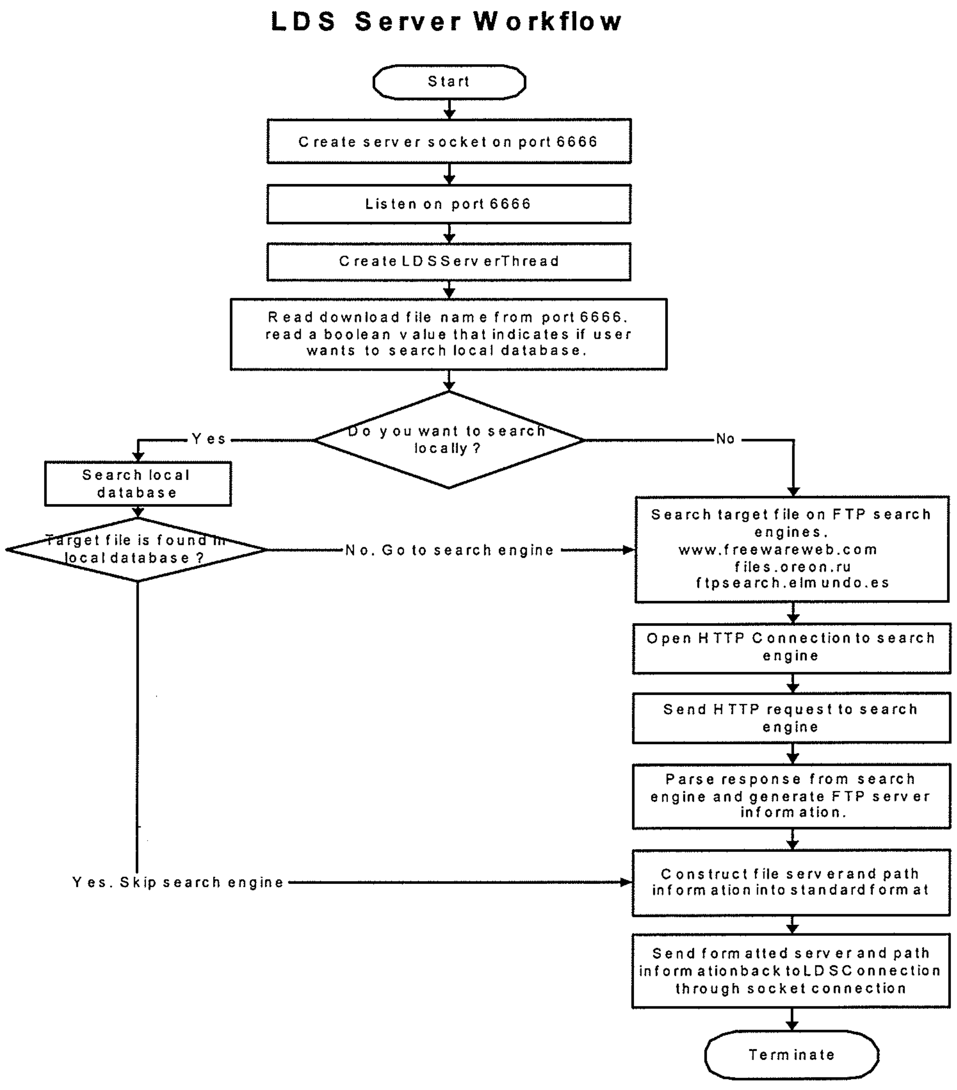

LDS Server Workflow
Start
Create server socket on port 6666
Listen on port 6666
Create LDSServerThread
Read download file name from port 6666. read a boolean value that indicates if user wants to search local database.
Do you want to search locally ?
Yes
No
Search local database
Target file is found in local database ?
No. Go to search engine
Search target file on FTP search engines. www.freewareweb.com files.oreon.ru ftpsearch.elmundo.es
Open HTTP Connection to search engine
Send HTTP request to search engine
Parse response from search engine and generate FTP server information.
Yes. Skip search engine
Construct file server and path information into standard format
Send formatted server and path information back to LDSConnection through socket connection
Terminate

FIGURE 4.3.2.1 LDS SERVER WORKFLOW

4.3.2.2. Search Engine Implementation

There are many kinds of FTP search engines available on Internet. When a user goes to a public FTP search engine, for example, www.freewareweb.com, they enter a file name and will get a response HTML page that contains all available FTP servers that they can download the target file from. In order to take advantage of public search engines, a search engine adaptor is defined that can send HTTP requests to the search engine, read response HTML page, and parse the HTML page for file server information. Then the adaptor sends file server information in a certain format back to the DFT client. Since each search engine returns an HTML page in a different format, it is necessary to define individual search engine adaptors for each engine.

For example, these are three separate search engine adaptors. They have similar structure but they process different HTML pages.

```
FTPSearchEngine.java
FileSearching.java
FreeWareWeb.java
```

Following diagram illustrates how search engine adaptor works.

FTP Search Engine Adaptor

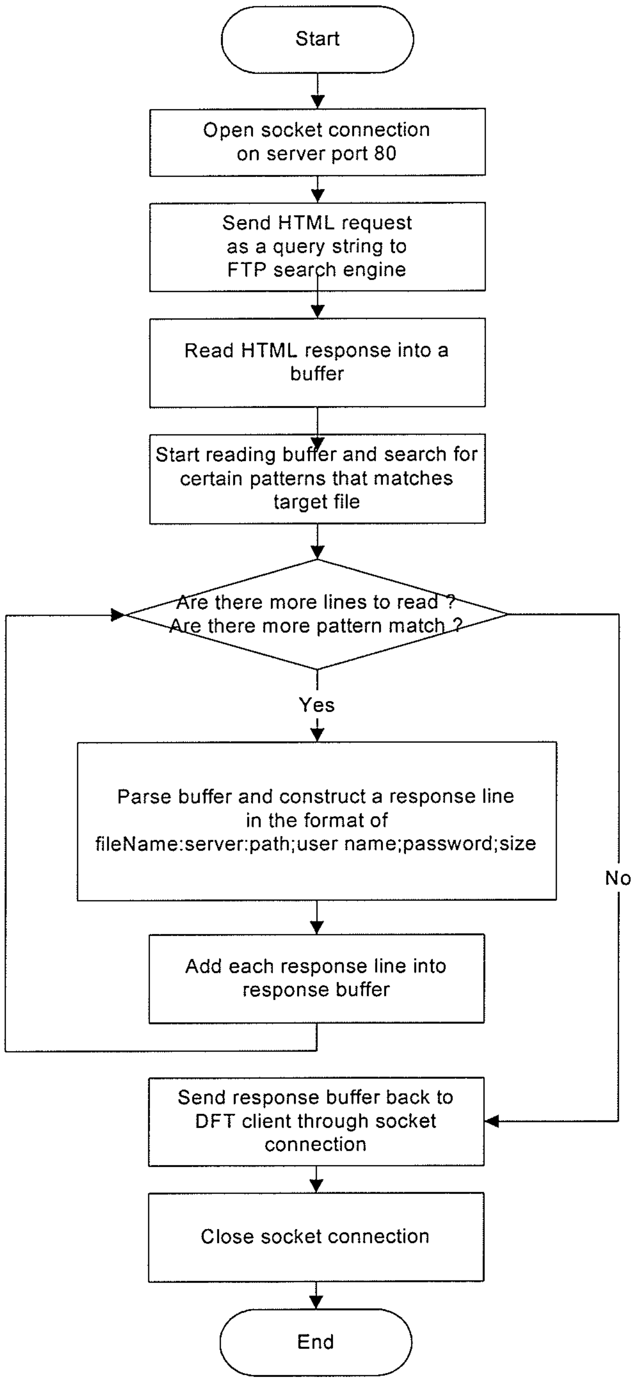

Start
Open socket connection on server port 80
Send HTML request as a query string to FTP search engine
Read HTML response into a buffer
Start reading buffer and search for certain patterns that matches target file
Are there more lines to read ? Are there more pattern match ?
Yes
Parse buffer and construct a response line in the format of fileName:server:path;user name;password;size
Add each response line into response buffer
No
Send response buffer back to DFT client through socket connection
Close socket connection
End

FIGURE 4.3.2.2 FTP SEARCH ENGINE ADAPTOR WORKFLOW

4.4. DFT System Module List

Table 4-1 lists all the DFT system modules including the packages that they belong to and their functions.

TABLE 4.4: SYSTEM UNIT MODULE LIST TABLE

Package	Module Name	Function
dft.gui		
dft.gui	SelectPanel.java	Allow user to select file server from a list
dft.gui	DFTFrame.java	The main DFT client window
dft.gui	SelectionTableModel.java	The table model for SelectPanel
dft.gui	SelectFrame.java	The window frame for SelectPanel
dft.gui	DFTFrame_AboutBox.java	Define the Help|About dialog box
dft.gui	DFTClient.java	The main DFT client starting point. Create DFTFrame.
dft.client		
dft.client	SubDownloadThread.java	Define the sub-download thread
dft.client	CommandException.java	Define the CommandException
dft.client	FileDownloader.java	Most of download logic is defined here. This module creates all download threads.
dft.client	ControlConnection.java	Contain logic to make control layer connection to FTP server and read response from FTP server.
dft.client	DataConnection.java	Contain logic to make data layer connection to FTP server download file form data port.
dft.client	DownloadThread.java	The thread to perform download
dft.client	FileInfo.java	Define target file and server information for each file server for downloading purpose.
dft.client	LDSConnection.java	Contain logic to make socket connection to LDS server.
dft.client	LogRecord.java	Utility module to write to log record
dft.server		
dft.server	FileWatcher.java	One of the search engine adaptor
dft.server	FTPSearchEngine.java	One of the search engine adaptor
dft.server	FileSearching.java	One of the search engine adaptor
dft.server	FreeWareWeb.java	One of the search engine adaptor
dft.server	OreonRu.java	One of the search engine adaptor

Package	Module Name	Function
dft.server	LDSServerThread.java	The thread to perform the file query and response task for LDS server.
dft.server	Veoda.java	One of the search engine adaptor
dft.server	Elmundo.java	One of the search engine adaptor
dft.server	LDSServer.java	Defines the LDS server class. LDS server creates LDSServerThread to complete file query tasks.
dft.util		
dft.util	LogManager.java	The logging utility class
dft.util	LDSConstants.java	Define all constants used in DFT project
dft.gensig		
dft.gensig	GenSig.java	Contain logic to generate digital signature
dft.versig		
dft.versig	SigChecker.java	Contain logic verify digital signature

4.5. Summary

In this chapter, the implementation of the DFT system was described. First, the implementation tools which include the Java language, Java Swing and JCreator (which is the Java IDE) are introduced. Then, the DFT packages and DFT client side and LDS server side implementations are described. How each part of the system works and how they interact with each other are explained in detail. A series of tests of the DFT system will be discussed next.

Chapter 5: Experiment and Data Analysis

Based on the DFT system design, architecture and implementation of all its functionalities described in previous chapters, a series of experiments will be carried out in this chapter. Through the experiment, the functionalities and features of the system described in the preceding chapters will be tested. The analysis of the experiment data will be done and a conclusion about the system will be presented at the end.

5.1. Experiment Goals and Design

5.1.1. Experiment Goals:

To improve the reliability and efficiency of the file transferring on Internet is the main motivation of the design of the DFT system. Testing the system's reliability and efficiency is the main part of the goals in this experiment. Since the system is designed for downloading from multiple servers and each with multiple threads, how the file transfer efficiency (speed) has been impacted by the multiple threads in different scenarios will be tested as the most important part.

5.1.2. Experiment Design:

5.1.2.1. Reliability Test:

Failure recovery is the indicator of the reliability. The system should be robust and be capable of detecting and recovering from different failures (described in Chapter 3)

In this test, the FTP server and the DFT client failures will be simulated and tested by stopping the FTP server and unplugging the DFT client network cable from it. The data of when these failures happen, how soon the DFT client detects these failures and how quickly it acts to switch is recorded.

In the server failure situation, the DFT client should be able to detect this disconnection from the server and switch its downloading task to the fastest server from which it is downloading. Behind the scene the DFT client will kill all the threads working on this server which include the threads for data connections and control connections. Then it will check all the other working servers that it has been downloading from and find the fastest server of them, then establish a new connection to this server, and continue downloading the file from the point where it was at when the former server failure happened.

In the client network failure, the client should be able to detect and prompt the user with a message and stop all the downloading tasks from all the threads.

5.1.2.2. Digital Signature Generation and Verification Test

Because the buttons of the both functionalities are on the GUI interface of the DFT client, the test is straightforward. The generator is used to generate a digital signature for downloaded file. The verifier is for the downloading user to verify the downloaded file. The functionality was only tested locally.

5.1.2.3. Efficiency Test

The efficiency test is to test how the system performs. As mentioned before, the system was designed for downloading from multiple servers, each with multiple threads. How the efficiency changes when the number of servers and the number of threads change is tested.

5.1.2.4. Auto Optimization Test

The test is bound to the efficiency test. When a server from which the DFT is downloading is becoming slow or halts because of traffic congestion or other reasons, the system should be capable of switching from this degraded server to another faster server.

5.2. Experiment Environment

In this section, the experiment environment including networks, download files, the requirements for running the DFT client and the LDS server are described in detail.

5.2.1. Networks

The Internet will be the network platform for the experiment. Since The DFT system is designed to download files from FTP servers scattered on the Internet, it must be able to download files from FTP servers running on the Internet. The network in the experiment will be the Internet rather than a limited local network. The DFT client will run on a Windows XP PC which is connected to the Internet through broadband Internet Connection (Shaw cable was the ISP when the experiment was being carried on). Figure 5.2.1 is the diagram of the networks (the local network and the Internet) for the DFT system test.

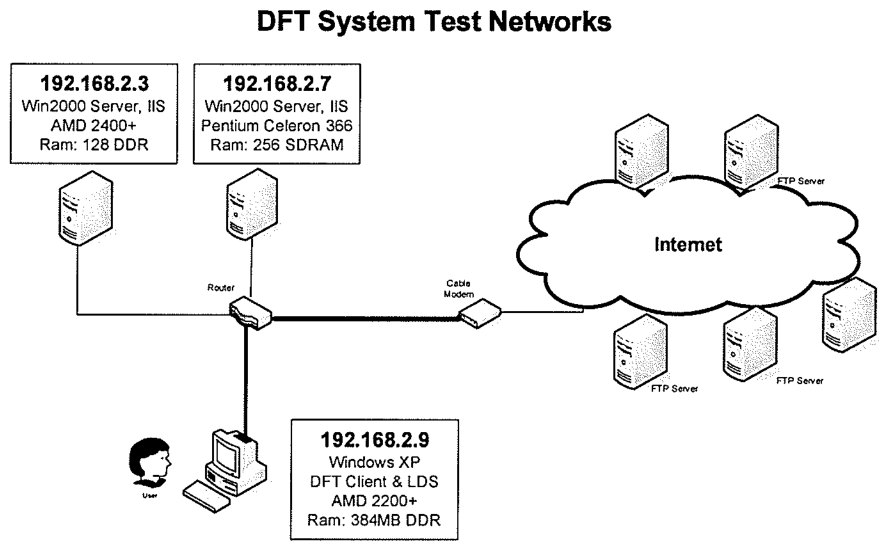


FIGURE 5.2.1 DTF SYSTEM TEST NETWORKS

5.2.2. Downloaded Files

Two groups of files are selected for the download tests. In the first group, two different files are used as the download test files in this experiment. They are mysql-4.1.12a-win32.zip (about 37.0 MB) and OOo_1.1.4_Win32Intel_install.zip (about 64.2MB). There are a lot of FTP servers holding these files across the Internet. Some of these FTP servers are chosen to download from.

1) mysql-4.1.12a-win32.zip. This ZIP file is the MySQL database server for windows. MySQL database server is the world's most popular open source database. This file is its official release of version 4.1.12a with the new windows installer as well as the Server Instance Configuration Wizard. The file size is about 37.0 MB. More info about MySQL database server can be obtained from its website [17].
2) OOo_1.1.4_Win32Intel_install.zip. This ZIP file is the OpenOffice.org suite for Windows. OpenOffice.org is both a multi-platform and multi-lingual office suite and an open-source project. It is compatible with all other major office suites, the product is free to download, use, and distribute. This file is its official release of version 1.1.4. The file size is 64.2MB. More info about OpenOffice.org can be obtained from its website [18].

The reason why these two files were selected is that they are popular on the Internet. The changing trends of the download speed on files less than 100MB can be obtained by repeating the download of the two files. The download tests on these two file are on the external FTP servers that holding them.

The other group of files chosen for the test is some files that were purposely made for the DFT system test. Their sizes are bigger than the above two files. Their sizes are about 200MB (200m.rar), 450MB (400m.rar), 680MB (600m.rar) and 960MB (900m.rar) respectively. They are put on the internal FTP servers and are downloaded repeatedly for checking the download efficiency change trends when the file sizes increase. These tests are only within the local network.

5.2.3. FTP Servers:

There are many FTP servers across the Internet. The user can get the FTP server list by using the build-in FTP server search engine function in the DFT client.

The FTP servers holding the FTP download files used in the experiments are primarily the official mirror sites of the two applications (mysql and openoffice) listed in their download pages.

A few local FTP servers are also setup on the local network. They are used to simulate network failures in the relevant experiments and for the tests of the bigger file download. Table 5.2.3 lists the external and internal FTP servers that are used for the download tests.

TABLE 5.2.3: FTP SERVERS USED IN THE EXPERIMENTS AND TESTS

File Name	Server Symbol	Server Address
mysql-4.1.12a-win32.zip	Anl	Mirror.mcs.anl.gov
	Banner	mysql.bannerlandia.com.ar
	Berlin	ftp.fu-berlin.de
	Ovh	mir1.ovh.net
	Sunsite	sunsite.informatik.rwth-aachen.de
	Wolf	ftp.fh-wolfenbuettel.de
OOo_1.1.4_Win32Intel_install.zip	uni-w	(1) ftp.uni-wuppertal.de
	Sunsite	(2) sunsite.informatik.rwth-aachen.de
	tu-bs	(3) openoffice.tu-bs.de
	Berlin	(4) ftp.fu-berlin.de
	Funet	(5) ftp.funet.fi
	uni-k	(6) ftp.uni-kl.de
	Arnes	(7) ftp.arnes.si
	Kulnet	(8) ftp.kulnet.kuleuven.ac.be
	e-tech	(9) ftp.e-technik.fh-muenchen.de
200m.rar, 400m.rar, 600m.rar, 900m.rar		
	3	192.168.2.3 (local FTP server)
	7	192.168.2.7 (local FTP server)

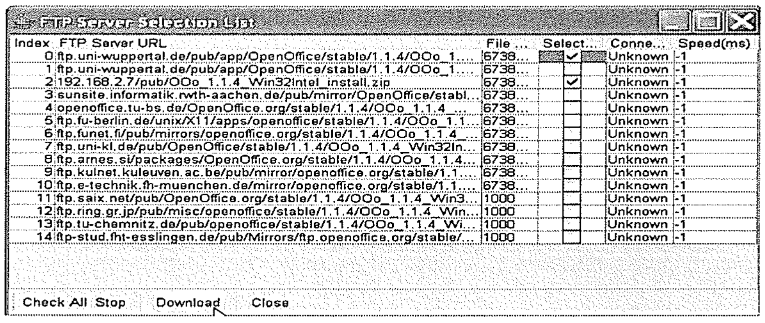
5.2.4. LDS Server and DFT Client:

Both the LDS server and DFT client run on a Windows XP PC which is connected to the local network. Its IP address is 192.168.2.9. The Java Runtime Environment installed is Java(TM) 2 Runtime Environment, Standard Edition 1.4.2_07. The Java IDE JCreator 3.5 is used to manage and launch the LDS server and DFT client. The LDS server runs under a DOS prompt. The DFT client is an intuitive and easy to use application with a friendly GUI interface. During the downloading test process, a lot of information is displayed on the GUI interface for the user. The server related information can be checked from the server DOS window.

5.3. Reliability test

5.3.1. Server Failure Recovery

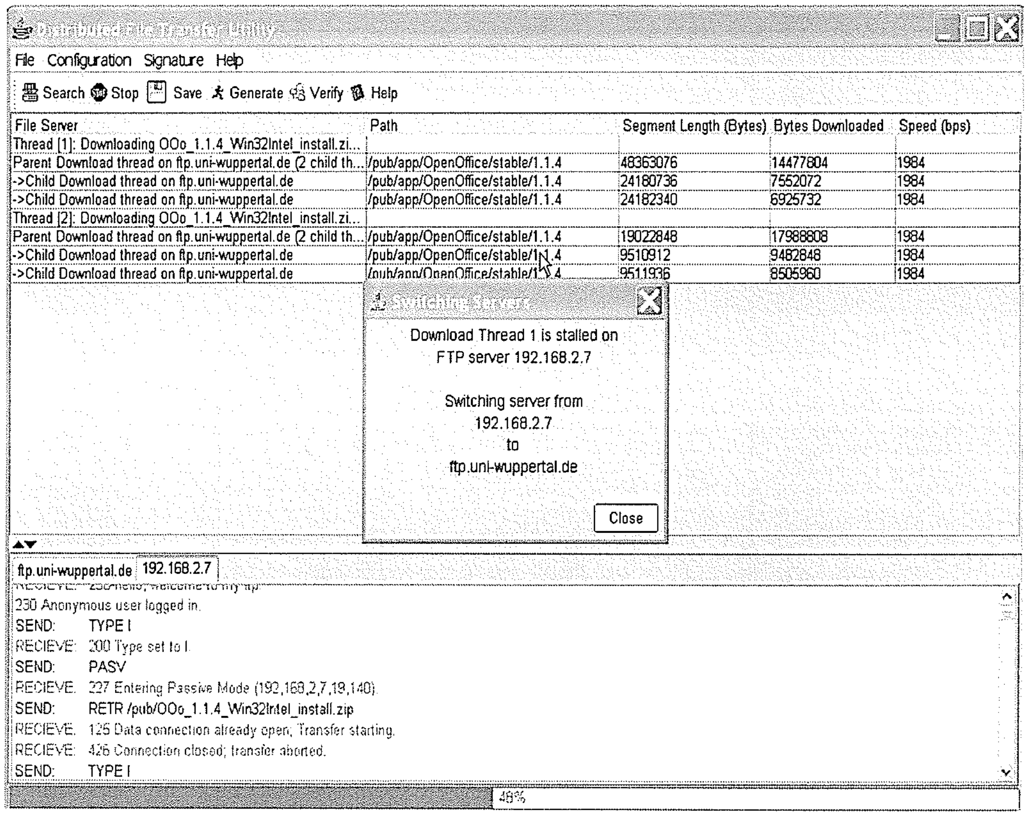
While the DFT client (IP: 192.168.2.9) is downloading from 2 FTP servers with one a local FTP server with IP address 192.168.2.7, and the other from the Internet with address ftp.uni-wuppertal.de, the local FTP server is stopped purposely to cause server failure (Figure 5.3.1a).



Index	FTP Server URL	File ...	Select...	Conne...	Speed(ms)
0	ftp.uni-wuppertal.de/pub/app/OpenOffice/stable/1.1.4/OOo_1....	6738...	✓	Unknown	-1
1	ftp.uni-wuppertal.de/pub/app/OpenOffice/stable/1.1.4/OOo_1....	6738...	☐	Unknown	-1
2	192.168.2.7/pub/OOo_1.1.4_Win32Intel_install.zip	6738...	✓	Unknown	-1
3	sunsite.informatik.rwth-aachen.de/pub/mirror/OpenOffice/stabl...	6738...	☐	Unknown	-1
4	openoffice.tu-bs.de/OpenOffice.org/stable/1.1.4/OOo_1.1.4_...	6738...	☐	Unknown	-1
5	ftp.fu-berlin.de/unix/X11/apps/openoffice/stable/1.1.4/OOo_1.1...	6738...	☐	Unknown	-1
6	ftp.funet.fi/pub/mirrors/openoffice.org/stable/1.1.4/OOo_1.1.4_...	6738...	☐	Unknown	-1
7	ftp.uni-kl.de/pub/OpenOffice/stable/1.1.4/OOo_1.1.4_Win32In...	6738...	☐	Unknown	-1
8	ftp.arnes.si/packages/OpenOffice.org/stable/1.1.4/OOo_1.1.4...	6738...	☐	Unknown	-1
9	ftp.kulnet.kuleuven.ac.be/pub/mirror/openoffice.org/stable/1.1....	6738...	☐	Unknown	-1
10	ftp.e-technik.fh-muenchen.de/mirror/openoffice.org/stable/1.1....	6738...	☐	Unknown	-1
11	ftp.saix.net/pub/OpenOffice.org/stable/1.1.4/OOo_1.1.4_Win3...	1000	☐	Unknown	-1
12	ftp.ring.gr.jp/pub/misc/openoffice/stable/1.1.4/OOo_1.1.4_Win...	1000	☐	Unknown	-1
13	ftp.tu-chemnitz.de/pub/openoffice/stable/1.1.4/OOo_1.1.4_Wi...	1000	☐	Unknown	-1
14	ftp-stud.fht-esslingen.de/pub/Mirrors/ftp.openoffice.org/stable/...	1000	☐	Unknown	-1

FIGURE 5.3.1A PICK UP THE INTERNAL SERVER AND AN EXTERNAL ONE

The DFT client detected the server problem and does the switch successfully.


File Configuration Signature Help
Search Stop Save Generate Verify Help
Close
ftp.uni-wuppertal.de
192.168.2.7
48%

File Server	Path	Segment Length (Bytes)	Bytes Downloaded	Speed (bps)
Thread [1]: Downloading OOo_1.1.4_Win32Intel_install.zi...				
Parent Download thread on ftp.uni-wuppertal.de (2 child th...	/pub/app/OpenOffice/stable/1.1.4	48363076	14477804	1984
->Child Download thread on ftp.uni-wuppertal.de	/pub/app/OpenOffice/stable/1.1.4	24180736	7552072	1984
->Child Download thread on ftp.uni-wuppertal.de	/pub/app/OpenOffice/stable/1.1.4	24182340	6925732	1984
Thread [2]: Downloading OOo_1.1.4_Win32Intel_install.zi...				
Parent Download thread on ftp.uni-wuppertal.de (2 child th...	/pub/app/OpenOffice/stable/1.1.4	19022848	17988808	1984
->Child Download thread on ftp.uni-wuppertal.de	/pub/app/OpenOffice/stable/1.1.4	9510912	9482848	1984
->Child Download thread on ftp.uni-wuppertal.de	/pub/app/OpenOffice/stable/1.1.4	9511936	8505960	1984

Switching Servers

Download Thread 1 is stalled on
FTP server 192.168.2.7

Switching server from
192.168.2.7
to
ftp.uni-wuppertal.de

```
230 Anonymous user logged in.
SEND:     TYPE I
RECIEVE:  200 Type set to I.
SEND:     PASV
RECIEVE.  227 Entering Passive Mode (192,168,2,7,19,140).
SEND:     RETR /pub/OOo_1.1.4_Win32Intel_install.zip
RECIEVE.  125 Data connection already open; Transfer starting.
RECIEVE:  426 Connection closed; transfer aborted.
SEND:     TYPE I
```

FIGURE 5.3.1B INTERNAL SERVER SWITCHED TO THE EXTERNAL SERVER

In this test, the time the FTP server stopped, the time the DFT client detected the server unavailability and the time the switch finished are recorded (Table 5.3.1).

TABLE 5.3.1: RESULTS OF TWO TESTS

Server Stop at	Client Detect at	Switch at	Overhead
Tue Jul 26 15:53:52	Tue Jul 26 15:53:56	Tue Jul 26 15:53:57	5 sec.
Tue Jul 26 15:50:21	Jul 26 15:50:23	Tue Jul 26 15:50:26	5 sec.

In both tests in the table, the period of time between the server stop and the DFT client finishes the switch is 5 seconds. The time between when the server stops and when the client detects and does the switch can vary greatly. These response times are affected by many elements such as the server response, the network traffic and the DFT client performance.

5.3.2. Network Failure

The network failure on client side was simulated by unplugging the network cable from NIC of the DFT client PC. The time from the unplugging to the client detecting this failure and popping out the error message is about 10 seconds. The time is recorded by a timer and includes a period of time from the client detection to the GUI error message popping up. The client is able to resume the download when re-downloading the file from the point the download was halted.

5.4. Signing a File and Verify a Signature

5.4.1. Signing a File

From the toolbars of the main window in the DFT client, clicking the 'Generate' button pops up a Windows Explorer. The file the user wants to sign (OOo_1.1.4_Win32Intel_install.zip in this experiment, Figure 5.4.1a) is thereby located.

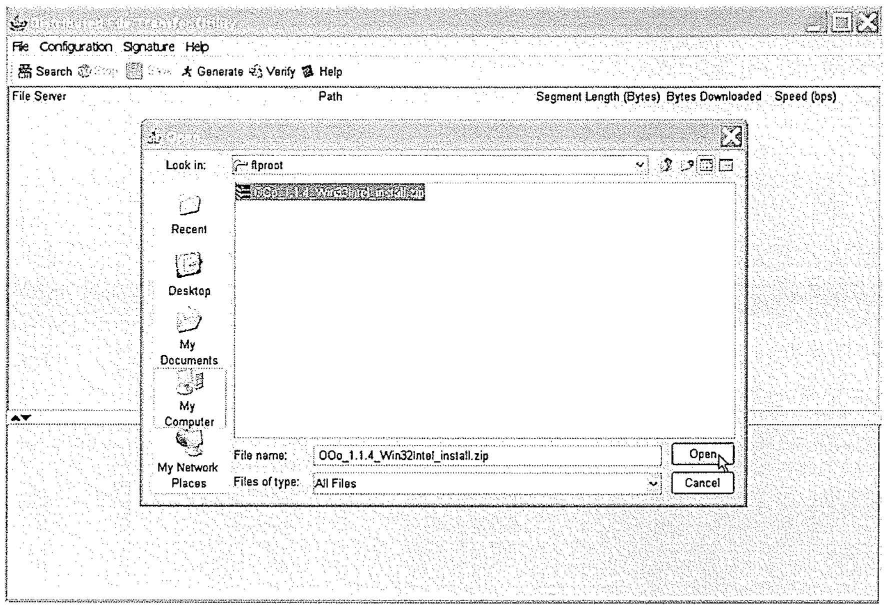

File Configuration Signature Help
Search Generate Verify Help
File Server Path Segment Length (Bytes) Bytes Downloaded Speed (bps)
Open
Look in: ftproot
Recent
Desktop
My Documents
My Computer
My Network Places
File name: OOo_1.1.4_Win32Intel_install.zip
Files of type: All Files
Open
Cancel

FIGURE 5.4.1A FIND AND CHOOSE THE FILE BEING SIGNED ON

The user then clicks the 'Open' button. A successful signature generate message is then displayed (Figure 5.4.1b).

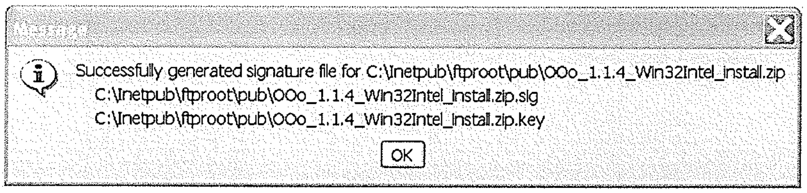
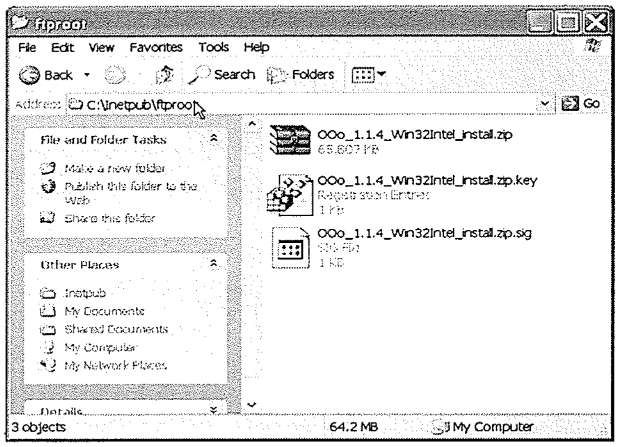

Successfully generated signature file for C:\Inetpub\ftproot\pub\OOo_1.1.4_Win32Intel_install.zip
C:\Inetpub\ftproot\pub\OOo_1.1.4_Win32Intel_install.zip.sig
C:\Inetpub\ftproot\pub\OOo_1.1.4_Win32Intel_install.zip.key

FIGURE 5.4.1B SUCCESSFUL SIGNATURE GENERATION MESSAGE

Two files, OOo_1.1.4_Win32Intel_install.zip.sig and OOo_1.1.4_Win32Intel_install.zip.key are generated in the same folder as the original file OOo_1.1.4_Win32Intel_install.zip (Figure 5.4.1c). These two files are the signature of the original file.



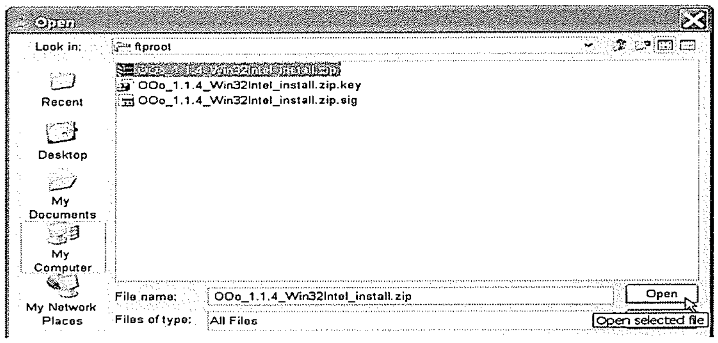
OOo_1.1.4_Win32Intel_install.zip
65,807 KB
OOo_1.1.4_Win32Intel_install.zip.key
Registration Entries
1 KB
OOo_1.1.4_Win32Intel_install.zip.sig
SIG File
1 KB

3 objects 64.2 MB My Computer

FIGURE 5.4.1C KEY AND SIGNATURE FILES ARE GENERATED

5.4.2. Verifying a Signature

When downloading a file, its digital signature files which are files with extensions of sig and key also need to be downloaded. In this case, they are OOo_1.1.4_Win32Intel_install.zip.sig and OOo_1.1.4_Win32Intel_install.zip.key. After finishing the download, the user puts the signature files with the original file in the same folder. Then from the main window of the system, click the 'Verify' button, and the Windows Explorer window pops up. The user then selects OOo_1.1.4_Win32Intel_install.zip (Figure 5.4.2a) and clicks the 'Open' button. The file is then verified against the public key and the signature.



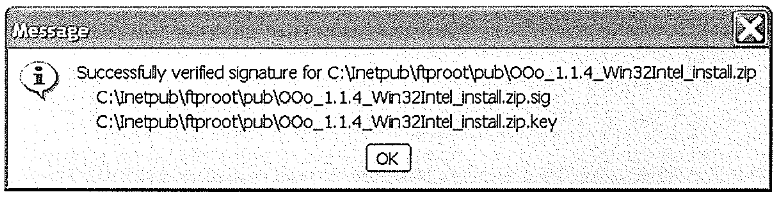
Look in: ftproot

OOo_1.1.4_Win32Intel_install.zip
OOo_1.1.4_Win32Intel_install.zip.key
OOo_1.1.4_Win32Intel_install.zip.sig

File name: OOo_1.1.4_Win32Intel_install.zip
Files of type: All Files

FIGURE 5.4.2A LOOKS IN THE FOLDER WHERE THE 3 FILES ARE

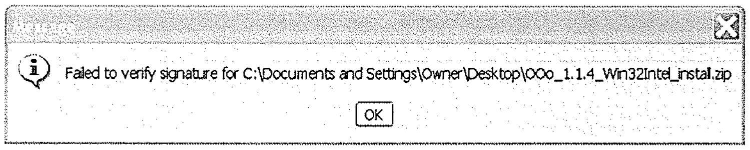
If it is successful, a message is displayed (Figure 5.4.2b).



```
Successfully verified signature for C:\Inetpub\ftproot\pub\OOo_1.1.4_Win32Intel_install.zip
    C:\Inetpub\ftproot\pub\OOo_1.1.4_Win32Intel_install.zip.sig
    C:\Inetpub\ftproot\pub\OOo_1.1.4_Win32Intel_install.zip.key
```

FIGURE 5.4.2B SIGNATURE IS VERIFIED SUCCESSFULLY

If the file OOo_1.1.4_Win32Intel_install.zip is replaced by a different zip file, the verifying procedure and an unsuccessful verification message will pop up (Figure 5.4.2c).



```
Failed to verify signature for C:\Documents and Settings\Owner\Desktop\OOo_1.1.4_Win32Intel_install.zip
```

FIGURE 5.4.2C FAIL TO VERIFY THE SIGNATURE

5.5. Efficiency Test on Files Less Than 100 MB

To test the efficiency of the system and get more accurate tendency results, a series of downloading tests from the selected external servers in different combinations for each file are performed. They can be categorized into the following groups:

1) To download with one, two and three threads from each server respectively;
2) To download from a combination of multiple servers with one, two and three threads respectively.

All the downloading tests on mysql-4.1.12a-win32.zip and OOo_1.1.4_Win32Intel_install.zip are from the selected servers outside the local network. That is, no local FTP servers are involved in this part.

The data of each download test is recorded. The downloading time when using a different number of threads to download from different servers is calculated.

5.5.1. Test Results

Table 5.5.1a is the test results on the file mysql-4.1.12a-win32.zip and table 5.5.1b is the test results on the file OOo_1.1.4_Win32Intel_install.zip.

TABLE 5.5.1A: TESTING DATA ON DOWNLOADING MYSQL-4.1.12A-WIN32.ZIP

Thread# / **Duration** / **Server(s)**	**One**	**Two**	**Three**	**Performance** 2 vs. 1 / 3 vs. 1
1 Server				
Anl	1 min 40 sec	1 min 1 sec	51 sec	+39% / +49%

Banner	3 min	1 min 21 sec	52 sec	+55% / +71%
Berlin	2 min 10 sec	1 min 20 sec	1 min	+38% / +54%
Ovh	7 min 10 sec	4 min 12 sec	2 min 42 sec	+41% / +77%
Sunsite	2 min 10 sec	1 min 30 sec	1 min 11 sec	+31% / +47%
Wolf	3 min 20 sec	2 min 21 sec	1 min 30 sec	+31% / +55%
1 Server Average	**3 min 15 sec**	**1 min 58 sec**	**1 min 23 sec**	**+39% / +59%**
2 Servers				
sunsite + banner	1 min 40 sec	1 min 2 sec	1 min 1 sec	+38% / +39%
anl +ovh	5 min 51 sec	2 min 51 sec	2 min 1 sec	+51% / +66%
sunsite + berlin	1 min 40 sec	1 min	1 min 1 sec	+40% / +39%
Wolf + ovh	5 min 50 sec	2 min 51 sec	1 min 42 sec	+51% / +71%
Wolf + anl	1 min 20 sec	1 min 21 sec	1 min 22 sec	-1%* / -3%*
Wolf + banner	3 min 20 sec	2 min	1 min 51 sec	+40% / +45%
Wolf + sunsite	2 min 30 sec	1 min 29 sec	1 min 2 sec	+41% / +59%
Wolf + berlin	3 min	2 min 10 sec	1 min 23 sec	+28% / +87%
2 Servers Average	**3 min 9 sec**	**1 min 51 sec**	**1 min 25 sec**	**+36% / +50%**
3 Servers				
Wolf+sunsite+anl	1 min 30 sec	1 min 2 sec	1 min 1 sec	+31% / +32%
Wolf+sunsite+ovh	1 min 50 sec	1 min 1 sec	1 min 11 sec	+45% / +35%
Wolf+sunsite+banner	1 min 30 sec	1 min 21 sec	1 min 2 sec	+10% / +31%
Wolf+sunsite+berlin	51 sec	1 min 11 sec	52 sec	-39%* / -2%*
Berlin+sunsite+anl	1 min 10 sec	1 min	52 sec	+14% / +26%
Berlin+sunsite+banner	1 min	1 min 1 sec	52 sec	-2%* / +13%
Berlin+sunsite+ovh	1 min 51 sec	1 min 21 sec	1 min 2 sec	+27% / +44%
Berlin+banner+anl	1 min	51 sec	51 sec	+15% / +15%
Berlin+banner+ovh	2 min 10 sec	1 min 30 sec	1 min 2 sec	+31% / +52%
banner+anl+ovh	2 min 20 sec	1 min 31 sec	1 min 15 sec	+35% / +46%
3 Servers Average	**1 min 34 sec**	**1 min 11 sec**	**1 min**	**+17% / +29%**
*: Downloading time with 2 threads or 3 threads is longer than that with 1 thread from each server				

TABLE 5.5.1B: TESTING DATA ON DOWNLOADING OOO_1.1.4_WIN32INTEL_INSTALL.ZIP

Thread# / Duration / Server(s)	One	Two	Three	Performance 2 vs. 1 / 3 vs. 1
1 Server				
Arnes	6 min 30 sec	3 min 11 sec	2 min 22 sec	+51% / +64%
e-tech	5 min 30 sec	2 min 20 sec	1 min 41 sec	+58% / +69%
Funet	3 min 50 sec	2 min 52 sec	2 min 31 sec	+25% / +34%
Berlin	4 min 20 sec	2 min 10 sec	1 min 41 sec	+50% / +61%
Kulnet	13 min 40 sec	7 min 41 sec	4 min 31 sec	+44% / +67%
Sunsite	5 min 10 sec	2 min 40 sec	1 min 51 sec	+48% / +64%
tu-bs	9 min 40 sec	3 min 51 sec	2 min 30 sec	+60% / +74%
Uni-w	5 min	3 min 22 sec	1 min 40 sec	+33% / +67%
Uni-kl	3 min 20 sec	2 min	1 min 41 sec	+40% / +50%
1 Server Average	**6 min 20 sec**	**3 min 21 sec**	**2 min 16 sec**	**+45% / +61%**
2 Servers				
Sunsite + tu-bs	3 min 30 sec	2 min 10 sec	1 min 51 sec	+38% / +52%
Sunsite + berlin	5 min 30 sec	2 min 30 sec	2 min 30 sec	+55% / +55%
Sunsite + funet	4 min	2 min 11 sec	2 min	+45% / +50%
Sunsite + uni-k	3 min 50 sec	2 min 11 sec	3 min 33 sec	+43% / +7%
Sunsite + arnes	3 min 41 sec	2 min 2 sec	1 min 52 sec	+45% / +49%
Arnes + kulnet	6 min 10 sec	3 min 21 sec	2 min 33 sec	+46% / +59%
Sunsite + uni-w	3 min 20 sec	2 min	1 min 41 sec	+40% / +50%
Uni-k + uni-w	2 min	1 min 42 sec	1 min 44 sec	+15% / +13%
2 Servers Average	**4 min**	**2 min 16 sec**	**2 min 13 sec**	**+41% / + 42%**
3 Servers				
Uni-w+sunsite+kulnet	5 min 30 sec	2 min 30 sec	2 min 2 sec	+55% / +63%
uni-w+sunsite+arnes	2 min 30 sec	1 min 40 sec	1 min 42 sec	+33% / +32%
uni-w+sunsite+uni-k	2 min 11 sec	1 min 32 sec	1 min 43 sec	+30% / +21%
uni-w+sunsite+berlin	1 min 40 sec	1 min 41 sec	1 min 40 sec	-1%* / 0
uni-w+tu-bs+berlin	2 min 1 sec	1 min 45 sec	1 min 40 sec	+13% / +17%
uni-w+sunsite+tu-bs	3 min	2 min	1 min 43 sec	+33% / +43%
Arnes+kulnet+e-tech	6 min 40 sec	3 min 11 sec	2 min 30 sec	+52% / +63%

sunsite+berlin+e-tech	1 min 40 sec	1 min 41 sec	1 min 41 sec	-1%* / -1%*
3 Servers Average	**3 min 9 sec**	**2 min**	**1 min 50 sec**	**+27% / +30%**
*: Downloading time with 2 threads or 3 threads is longer than that with 1 thread from each server				

5.5.2. Experiment Analysis

After studying the data collected in the above two tables, it is easy to find that the relations between efficiency of the system and the number of servers and/or the number of threads used to download from each server.

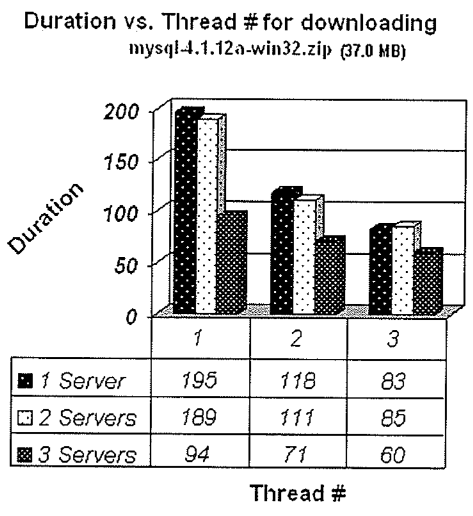
5.5.2.1. Single server Download – one vs. multiple threads

Compared with multiple threads, downloading from a single server with one thread is the least efficient. The data in the "1 server section" of each table above shows that download from a server with 2 threads or 3 threads improves the download speed greatly. The average improvement rate is 39% with 2 threads, 59% with 3 threads from 1 server for downloading mysql-4.1.12a-win32.zip. The rates are 45% better with 2 threads and 61% with 3 threads from 1 server for downloading OOo_1.1.4_Win32Intel_install.zip. So a conclusion can be reached that downloading from a single server with multiple threads is more efficient than using only one thread.

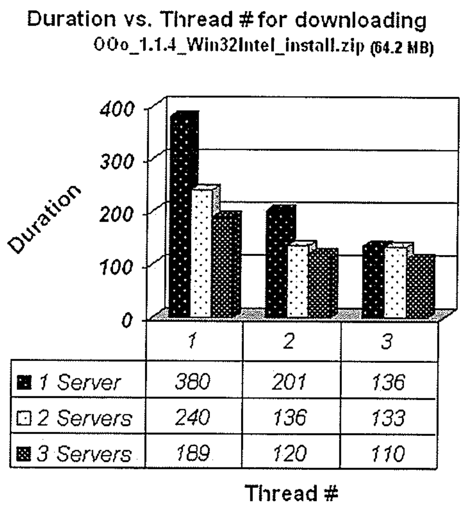
5.5.2.2. Multiple servers Download – one thread vs. multiple threads

From 2 and 3 server selections of the above tables, the download speed from different combination of selected servers with 2 or 3 threads is generally faster than that with just one thread. The average improvement rate is 36% with 2 threads, 50% with 3 threads from 2 servers, 17% and 29% from 3 servers for downloading mysql-4.1.12a-win32.zip. The rates are 41% better with 2 threads, 42% with 3 threads from 2 servers, 27% and 30% from 3 servers for downloading OOo_1.1.4_Win32Intel_install.zip. In general, using 3 threads makes the download speed even faster than using 2 threads.

Figure 5.5.2.2 shows how the download time changes with download thread numbers when using 1 thread, 2 threads and 3 threads to download the two files. Axis x is the change of thread numbers and axis y is the change of download time.



	1	2	3
■ 1 Server	195	118	83
□ 2 Servers	189	111	85
▨ 3 Servers	94	71	60



	1	2	3
■ 1 Server	380	201	136
□ 2 Servers	240	136	133
▨ 3 Servers	189	120	110

Figure 5.5.2.2 Duration vs. Thread Numbers

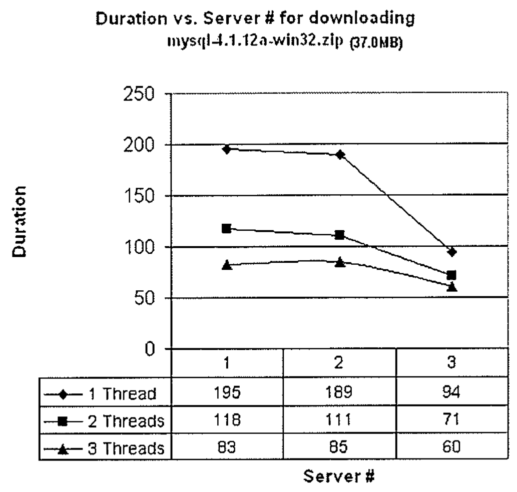
The shapes in the above two graphs are not the same, though ideally they should be. This is because during the download procedure, the download speeds were not constant but were affected by some elements such as the discrepancy of the test speed and real download speed, the overhead that was caused by random access file mechanism.

5.5.2.3. One Server vs. Multiple Servers

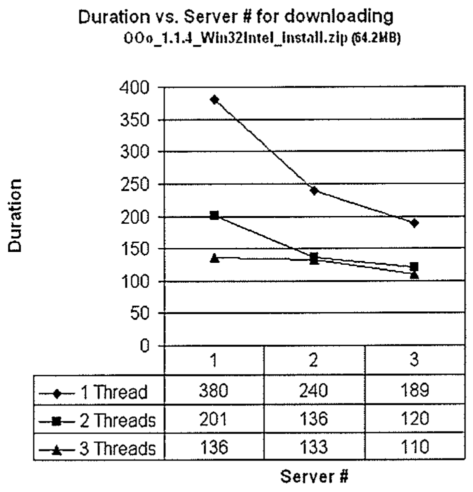
As discussed in the previous sections, the advantage in downloading from multiple servers over just one server is its robustness and improvement in reliability. In other words, when downloading from multiple servers, if one or some of the servers fail to provide the file download, the system can still finish the download from other working server(s) unless all the servers are down.

In the experiment, only downloads from one server, two-server and three-server combination are tested. For each download, 1 thread, 2 threads and 3 threads are tested. As far as the limited test data tells, on average, downloading from 3 servers is faster than downloading from two servers and downloading from two servers is faster than downloading from one server, though this is not always true for a specific download.

Figure 5.5.2.3 shows how the download time changes with download server numbers when using 1 thread, 2 threads and 3 threads to download the two files. Axis x is the change of server numbers and axis y is the change of download time.



Server #	1	2	3
1 Thread	195	189	94
2 Threads	118	111	71
3 Threads	83	85	60



Server #	1	2	3
1 Thread	380	240	189
2 Threads	201	136	120
3 Threads	136	133	110

FIGURE 5.5.2.3 DOWNLOAD TIME VS. SERVER NUMBERS

Once again, the shapes of these two graphs are different, though they indicate the same trend. The overhead that was caused by the random access file mechanism and the difference between the test speed and the real download speed are the main contributors to this discrepancy.

5.5.2.4. Elements Affecting Download Speed

1) Not all the FTP servers have the same performance. The servers with lower performance make the whole download period longer, and the servers with higher performance will make the whole download period shorter.

2) The task allocation among the FTP servers is based on the detected speed before the downloading starts by downloading 10k of data from each server. Thereafter if the download speed does not change much, then all downloads from different servers should finish at almost the same moment.

If the server's speed increases, it will make the download faster. If the server's speed decreases, it will make the download slower.

5.5.2.5. Low Speed Server Switch

The system has a functionality to monitor all the download threads' speed. It employs an independent thread called SpeedTimer to check the download threads' speed every 10 seconds. If the speed which is the total download amount of every 10 second span is divided by 10 seconds is lower than half of the original detected speed, the slower download thread is stopped and switches its task to a server and carries on the downloading from the new server.

5.6. Efficiency Test on Bigger Files

To test how the file sizes affect the download speed in the DFT system, the tests on the bigger files with sizes of 200MB, 450MB, 680MB and 960MB respectively were carried out. All the downloading tests on bigger files were within the local network and these downloads were from 2 local FTP servers with IP 192.168.2.3 and 192.168.2.7. The download processes were done repeatedly and the average download time was calculated.

5.6.1. Test Results

TABLE 5.6.1: TESTING DATA ON DOWNLOADING BIG FILES

Thread# / Duration	One	Two	Three	Performance 2 vs. 1 / 3 vs. 1

Servers				
1 Server: (3)				
200m.rar	58 sec	50 sec	49 sec	+14% / +16%
450m.rar	2 min 07 sec	2 min 03 sec	2 min 0 sec	+0.03% / +0.06%
680m.rar	4 min 17 sec	3 min 56 sec	4 min 10	+8% / +3%
960m.rar	4 min 26 sec	4 min 58 sec	6 min 07 sec	-10% / -38%
1 Server: (7)				
200m.rar	3 min 10 sec	3 min 08 sec	3 min 07 sec	+1% / +2%
450m.rar	6 min 52 sec	7 min 03 sec	7 min 12 sec	-3% / -5%
680m.rar	10 min 09 sec	10 min 02 sec	11 min 0 sec	+1% / -8%
960m.rar	14 min 56 sec	15 min 05 sec	15 min 25 sec	-1% / -3%
2 Servers: (3)+(7)				
200m.rar	2 min 04 sec	1 min 28 sec	1 min 55 sec	+21% / +7%
450m.rar	5 min 52 sec	5 min 51 sec	6 min 07 sec	+0% / -4%
680m.rar	7 min 11 sec	8 min 59 sec	9 min 38 sec	-16% / -34%
960m.rar	13 min 45 sec	14 min 22 sec	21 min 20 sec	-4% / -54%

5.6.2. Experiment Analysis

The data in the above table shows how the download speed changes with 1 thread, 2 threads and 3 threads on files with different sizes. Like the tests on smaller files, the increase of server number means the increase on the reliability. But the multi-thread download speed is not always increased when the download size gets bigger.

When downloading a file with size less than 680MB, the 2-thread download speed from 1 server generally increases the single thread download, but this increase is smaller than that when downloading the mysql and OpenOffice in the smaller files tests above. The 3-thread download usually takes more time than using 1 or 2 threads. When downloading the file of 960MB using either 2-threads or 3-threads takes more time than a single thread download, independent of whether the download is from a single server or 2 servers.

The main reason why the multi-threaded download does not improve the download speed on big files like 960MB is that the overhead caused by random access file counters the advantage that the multiple-threads would bring. In the random access mechanism, the file pointer does not move in a sequential but random order to read or write. Before reading or writing, each download thread needs to move the file pointer to the proper place before it can start to operate. The bigger the download file, the more time needed to do this movement. In addition, this test was only local, whereas DFT presumably would work better in a WAN.

5.7. Summary

In this section, a series of experiments and tests are designed and implemented. Through the tests and experiments, the system's reliability, efficiency, and other features are tested. As far as the results show, the system can recover from failures, make and verify the signatures. It can improve the download speed by using multiple threads to download from multiple servers concurrently on files of size less than certain sizes (< 100MB).

Chapter 6: Conclusions and Future Work

Reliability and efficiency are the two most important concerns of file transfer over the Internet. This thesis provides the design and implementation of a mechanism for the reliable, efficient file transfer from FTP servers scattered over the Internet. The authenticity, data integrity and failure recovery philosophy of the file transfer are also described and implemented to a degree.

Distributed file transfer increases the reliability of the file transfer over the Internet. Generally speaking, the more servers the file is being downloaded from simultaneously, the higher the reliability. The file is divided into different segments and each server is assigned a segment according to its transfer speed. When one server stops during this time, its task will be switched to another server. The reliability of the file transfer increases when the number of servers increases.

Efficiency is another important concern regarding file transfer over the Internet. Here, download speed is its indicator. Through the experiments and tests, it is noticed that when the download file size is less than a certain size, the download speed can be improved by multiple-thread download, independent of whether the download is from a single server or from multiple servers. However, this improvement gets weaker and diminishes with the increase of the file size. Many elements can impact the download speed. These elements include the network connection speeds, the client hardware and software, the servers' hardware and software and especially the download file size. The overhead generated by multiple servers and multiple threads should be taken into account when assessing the improvement. The download speed does not always increase with the increase of the server number and thread number. The multi-threaded download speed degrades when the download file size is over a certain point.

Future work on the DFT system may include:

1) The user can configure whether the DFT system should switch a stalled server or keep monitoring the server's speed. If they decide to keep monitoring the server's speed, when the stalled server recovers, the download from this server should resume. This is meaningful when the download file size is big or does not have extra mirrors to switch to.

2) Let each download thread have a separate 'saveas' file rather than share one random access file with others. The system joins all these separate files together when all downloads finish. This may reduce the overhead caused by the random access mechanism on a huge file.

3) For the digital signature, the public key is currently assumed to be stored on the download ftp servers and should be downloaded as a separate file. For the further development, it can be implemented in this way: store the public keys on the LDS server. When the file and its signature are downloaded, the DFT system will get

the corresponding public key from the LDS server and then verify the signature automatically.

4) Improve the DFT system's intelligence so that it is smart enough to decide what FTP servers are the best for the user's download and launch proper number of threads according to the download file size.

References

[1] T. Parker, Teach Yourself TCP/IP in 14 Days, 2nd Ed., USA: Sams Pub., 1996.

[2] R. Zakon, "Hobbes' Internet Timeline v8.0", [Online document] Jan. 2005, Available at HTTP: http://www.zakon.org/robert/internet/timeline/#2000s

[3] D. Reilly, "Inside Java: The Java Programming Language", [Online document] Nov. 1992, Available at HTTP: http://www.javacoffeebreak.com/articles/inside_java/insidejava-nov99.html

[4] RFC959, "Internet RFC/STD/FYI/BCP Archives", [Online document], Advameg, 2003, Available at HTTP: http://www.faqs.org/rfcs

[5] FlashGet, amazesoft, 2005, Available at HTTP: http://www.amazesoft.com

[6] Internet Download Manager, Tonec, 2005, Available at HTTP: http://www.internetdownloadmanager.com

[7] SnapFiles, WebAttack, 2005, Available at HTTP: http://www.snapfiles.com/

[8] M. Miller, Discovering P2P, Alameda: SYBEX, 2001.

[9] eDonkey, MetaMachine, 2005, Available at HTTP: http://www.edonkey2000.com/

[10] Bittorrent, 2005, Available at HTTP: http://www.bittorrent.com

[11] GetRight, Headlight, 2005, Available at HTTP: http://www.getright.com

[12] X. Fang, "Reliable File Transfer on the Internet Using Distributed File Transfer (DFT)", University of Manitoba, 2000

[13] Bittorrent, 2005, Available at HTTP: http://www.bittorrent.com

[14] R. Whittle, "Cryptography for encryption, digital signatures and authentication", [Online document], 1996 December 19, Available at HTTP: http://members.ozemail.com.au/~firstpr/crypto/index.html

[15] Sun, "J2SE: Java Foundation Classes (JFC) Overview", [Online document], Available at HTTP: http://java.sun.com/products/jfc/overview.html

[16] JCreator, "A Java IDE", Xinox, 2005, Available at HTTP: http://www.jcreator.com

[17] Mysql, "Open Source Database", MySQL AB, 2005, Available at HTTP: http://www.mysql.com

[18] OOo, "Open Source Office Suite", Available at HTTP: http://www.openoffice.org

[19] C. Leiden and M. Wilensky, TCP/IP for Dummies, 5th Ed., New York: Wiley, 2003.

[20] D. Comer, Internetworking with TCP/IP – Vol.1, 3rd Ed., Prentice Hall, 1995.

[21] Wikipedia, "Digital Signature", Available at HTTP: http://en.wikipedia.org/wiki/Digital_signature

[22] R. Cadenhead and L. Lemay, Sams Teach Yourself Java 2 in 21 Days, 4th Ed., USA: Sams, 2004.

[23] S. Burnett and S. Paine, RSA Security's Official Guide to Cryptography, Berkeley: McGraw-Hill, 2001.

[24] A. Williams, Java2 Network Protocols Black Book, Scottsdale: Coriolis, 2001.

[25] I. Horton, Beginning Java 2, SDK 1.4 Ed., Birmingham: Wrox, 2003.

[26] C. Horstmann and G. Cornell, Core Java™ 2 Vol. I - Fundamentals, 7th Ed., Prentice Hall, 2004.

[27] C. Horstmann and G. Cornell, Core Java™ 2 Vol. II - Advanced Features, 7th Ed., Prentice Hall, 2004.

[28] M. Tulloch, IIS 6 Administration, Osborne/McGraw-Hill, 2003.

Appendix

A. RFC FTP Protocol, [Online document], Available at FTP:
ftp://nic.merit.edu/documents/rfc/rfc0959.txt

B. FTP Command List, [Online document], Available at HTTP:
http://www.nsftools.com/tips/RawFTP.htm

C. FTP Error Codes Explained, [Online document], Available at HTTP:
http://www.the-eggman.com/seminars/ftp_error_codes.html

100 Codes	**The requested action is being taken. Expect a reply before proceeding with a new command.**
110	Restart marker reply.
120	Service ready in (n) minutes.
125	Data connection already open, transfer starting.
150	File status okay, about to open data connection.
200 Codes	**The requested action has been successfully completed.**
200	Command okay.
202	Command not implemented
211	System status, or system help reply.
212	Directory status.
213	File status.
214	Help message.
215	NAME system type. (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 (file transfer, abort, etc.).
227	Entering Passive Mode
230	User logged in, proceed.
250	Requested file action okay, completed.
257	"PATHNAME" created.

300 Codes	**The command has been accepted, but the requested action is being held pending receipt of further information.**
331	User name okay, need password.
332	Need account for login.
350	Requested file action pending further information.
400 Codes	**The command was not accepted and the requested action did not take place. The error condition is temporary, however, and the action may be requested again.**
421	Service not available, closing control connection. (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 Codes	**The command was not accepted and the requested action did not take place.**
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	User not logged in.
532	Need account for storing files.
550	Requested action not taken. File unavailable (e.g., file not found, no access).
552	Requested file action aborted, storage allocation exceeded
553	Requested action not taken. Illegal file name.