University of Pittsburgh

Computer Science

CS 2520 - WANs

Simple File Transfer Protocol

Project Report

Authors:
Debashis Ganguly
Darshan Shetty

Supervisor:
Dr. Taieb Znati

April 23, 2015

Abstract

The design and implementation of a *Simple File Transfer Protocol*, allows the user to issue commands and transfer files between systems over an error prone transport channel. Our application mainly has 3 components: Client which is the active component of the system and is exposed to the user, File Server which services the client and usually be the listener and the Name Server which holds the list of available file servers and is responsible for the connection between the client and the server. The project exposes us to different aspects of computer and network programming like client/server paradigm, concept of layering and interfaces, and a better understanding of error and flow control implementation. In our implementation of the SFTP, we have used UDP (User Data-gram Protocol) as the control and the data channel, over which we have implemented Go-back-N protocol and CRC for error and flow control. Replicated name servers are implemented for the robustness of the system and each server can handle multiple clients at a time. Injected bit error into the data packet and emulated the congestion probability by dropping the packets in order to study the performance of the FTP.

Contents

1 Introduction

The document in review serves as a project report for implementation of *Simple File Transfer Protocol (SFTP)* as part of the project towards completion of the course CS 2550 in Spring, 2015. This design document details components and sub-components participating and inter-playing in the final system implemented and the layers of the protocol under discussion. It also acts as a high-level guide for the implementation and realization of the project. Not only that it details out high-level test strategies, statistical approaches to validate the correctness and quantify the effectiveness in terms of performance metrics for the project.

The goal of the project, that this document deals with, is to walk through the design, implementation, test and analyze a *Simple File Transfer Protocol (SFTP)*. This protocol enables users to issue commands to and interact with the application to effectively share files between two end-points of a distributed system. The word *effectiveness* captures different aspects and dimensions of the protocol namely reliability, availability, fault tolerance, exception handling etc. As end user functionality, the system, upon realization, will ease navigation and access of file and directory structure by the users before they can send a file from local to remote machine or request for copying a file from remote to local machine. It is to be noted that local system is termed as *Client* and whereas the remote machine is known as *File Server*. The directory and file server in concern is abstracted by the name-space of access provided by *Andrew File System (AFS)*. The proposed protocol will also ensure error detection using *Cyclic Redundancy Check (CRC)* and error and flow control using *Automated Repeat Request (ARQ)*. So as a pre-requisite to the design, the document assumes a connected set of end-systems to form and support a distributed application as a platform for implementation of the protocol.

Notations and Abbreviations

Below are list of useful Notations and Abbreviations which are used throughout the document interchangeably with their full form.

SFTP: *Simple File Transfer Protocol*
NS: *Name Server*
FS: *File Server*
C: *Client*
UI: *User Interface*
CFM: *Control and File Management*
DD: *Data Delivery*
TC: *Transprt Channel*
ACK: *Acknowledgment*
NAK: *Negative Acknowledgement*
CRC: *Cyclic Redundancy Check*
ARQ: *Automated Repeat Request*
FSname: *Logical name of the file server*
n_T: *Number of try*

TCP: *Tranmission Control Protocol*
UDP: *User Datagram Protocol*
T_{Out}: *Time out period*
MTU**:** *Maximum Transmission Unit*
S_w**:** *Window Size*
p_{Err}**:** *Probability of packet in error*
P_C**:** *Percentage Congestion*
F_{NS}**:** *Fully qualified path for Name Server file*
S_{LF}**:** *Fully qualified path for Statistical Analysis log file*

2 System Overview

This section identifies the components of the proposed system and defines them by their functionality without getting into details of their individual architecture or the interactions between them. It also briefly talks about the layering of SFTP upon which aforementioned components act.

2.1 System Components

The whole system can be visualized by the below major components.

Name Server (NS):

This application component runs on a set of end systems of the distributed network and these nodes will be providing services to the file servers to register themselves to the system and services to the client to query the details of the file servers.

File Server (FS):

This is the piece of application that runs on a set of end systems of the distributed application which are going to be known as File Server. They will be responsible for providing services to the Clients(s) to issue commands to navigate and access AFS and will be serving requests from Client(s) to send and/or receive files via network.

SFTP Client (C):

This component is the overall consumer of the distributed system. They consume the service provided by NS and FS. This part of application will be exposed to end users to issue several requests using known commands to access AFS, copy files etc.

2.2 Protocol Layering

The components of the system takes up separate role with a clearly differentiable sets of functionality. They participate in the protocol layering and integrate as a whole system. The high-level layers of the protocol can be conceptualized as below.

User Interface (UI):

This is the top most or outer most layer which provides a platform to the users to interact with the system. The UI defines a set of commands which are known as form of knowledge document or help file to the user. This layer passes the user to the next file management layer to interpret. The UI is implemented in the SFTP Client component. *We have implemented the command-line interface (CLI) for user to issue commands for effective file transfer between the machines.*

Control and File Management (CFM):

The CFM layer is implemented in both the clients and server systems. The high level functionality of CFM can be visualized in two dimensions- firstly interpret the commands issued by users at UI and translate them to the below layer by invoking certain primitives and secondly handling requests for directory navigation and file share thus the overall responsibility of file management. This is also responsible for handling exceptions encountered by the below layer and pass as user friendly message to the upper layer to ensure smooth execution of the system.

Transport Channel (TC) and Data Delivery (DD):

This component constructs the fundamental or base of the whole protocol and thus is found in all systems or nodes participating in distributed architecture. It is responsible for transferring control and data over network in fundamental and is also extended to ensure reliability of information exchange between client and server by providing error detection, flow and congestion control.

3 System Architecture and Layering

This section explains functionality and architecture of aforementioned components of the system and layered architecture of SFTP in details.

3.1 Details of System Components and sub-components

3.1.1 Name Server

As mentioned in Section 2, this is the top most component of the application that is initialized in the beginning or in other words the system bootstraps by initializing the NS first.

The client is aware of the logical name of FS and queries NS to resolve the logical name of FS to obtain IP address and Port Number before establishing connection with file server. So, before any client can start communicating with File Server, File Server register itself with NS. The Name Server maintains a hash table in its primary memory. The hash is keyed by logical name of the File Server. The value of the hash entry is a data structure as shown in following figure containing IP Address, Port Number of the FS and number of clients it is currently servicing.

The dynamic array [3] is maintained within a Service Table residing in primary memory of NS. Currently the service table will contain only entry for SFTP Service though it can be extended in future to have entries for different services like Printing Service etc.

Design Principle of NS:

In a distributed system, simplistically there can be a single Name Server for FS to register and Client to resolve query. As a system designer we need to think of robustness and fault-tolerance of the system such that even if the NS server stops performing due to some system fault, the system as a whole shouldn't collapse but continue performing. To ensure the availability of NS, we need to have back-up server running same component

of the NS and maintaining the same list of registered servers in sync with the primary NS. This can be achieved by two means as below.

Stand-by NS: In this approach, there is a Master NS which is the primary NS and there is Slave NS which acts as secondary or back-up NS. Only the master server is active at any instance and the other one is waiting in passive mode. The Master NS handles all the requests for resolving the name of the server and registering new servers, while the back-up NS is passive and listens to health signal sent by the Master NS. At a regular interval, Master sends a 'I am alive' signal to the Slave. All resolve requests from client is served by the Primary NS. Only in case of failure of Primary NS, the Slave does not receive the health signal from the Master for a certain time out period and then it takes over the role of Master or Primary NS.

This method is used when the load on the NS is not too heavy and a single NS can handle all the requests from client and server. Moreover, if we want to utilize the Secondary NS to perform other tasks independently at its idle time without bothering register or resolve requests.

Replicated NS: In this approach both the NS server shares the load of register and resolve requests equally and so both the servers are required to register their IP and port number at 'Name_Server_Info.txt' on the AFS and maintain the synchronised hash table for list of FS. Client and FS read both of the IPs and Port numbers from the file 'Name_Server_Info.txt' and maintains it at cache memory. Clients or FS randomly pick one of the NS for service from their list and request accordingly. Upon failure of one NS, Client and FS remove its entry from the cache and redirect all the requests henceforth to the other one.

This method is effective when the load of register and resolve requests are very high as both the servers share the load equally based on the efficiency of random function at client and FS.

Implementation Choice: In our project we are choosing Replicated NS over Stand-by NS as following the discussions and arguments made above from the perspective of scalability of system. The details of the steps of bootstrapping NS can be found in Section 3.2.

3.1.2 File Server

As discussed in Section 2, File Server is the second component in hierarchy which gets initialized after Name Server. File Server is accountable for servicing requests from Clients for file access and file share.

The software component that runs on File Server bootstraps the system. File Servers register itself to Name Server so that they are to be known and accessible by clients to establish connection. A File Server can serve multiple clients at a given instance. The number of concurrent clients is a configurable parameter which is decided after load testing and stress testing.

Implementation Choice of FS:

In a distributed system, simplistically there can be a single FS to serve all Clients. This hinders the scalability of the system as at a given point of time only a fixed number of clients can be served by a single FS. Even as a system designer we need to think of robustness and fault-tolerance of the system such that even if the FS stops performing due

to some system fault, the system as a whole shouldn't collapse but continue performing and servicing all the clients.

To improve scalability and ensure the availability of FS, we need to have multiple back-up servers running same component of the FS. All these servers have access to the same file repository in AFS and known by a same logical name to provide transparency and abstraction to the client. Clients are only aware of the logical naming of FS, it is the responsibility of NS to resolve the name and ensure effective utilization of the list of FS resources.

3.1.3 Client

Client is the active component of the system which run on several machines. These are the systems actually exposed to the end users of the application.

Clients are aware of the logical naming of file server and request to NS for resolve to acquire IP and Port Number before establishing any connection with FS. Clients requests the FS for a certain number tries and wait for a timeout period for response. Upon failure from FS to send any response, client component pro-actively look for the NS to obtain another FS to connect with.

3.2 Component Life Cycle and Interplay

3.2.1 Bootstrapping Name Sever

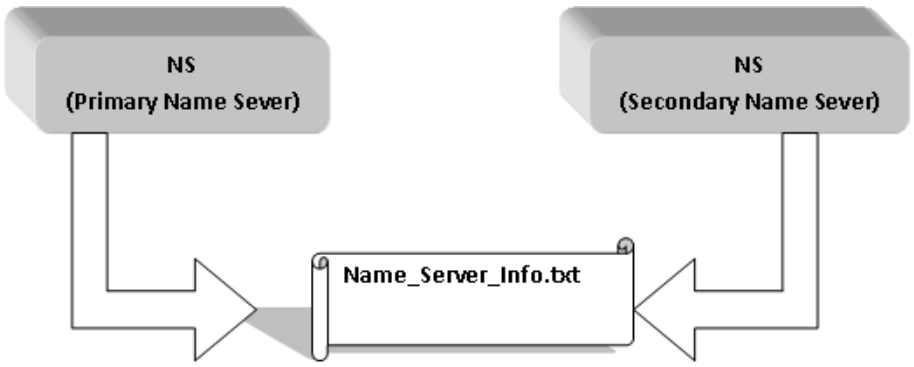

Figure 1: Bootstrapping Name Server

Step 1: The software component for NS is run to initialize NS.

Step 2: As we are not hard-coing IP address and Port Number of any component to improve portability, NS makes a system call to get it's IP and port number. It then writes the same into the well known file 'Name_Server_info.txt' in AFS, so the file is can be accessed by the FS and Clients.

Step 3:If there is a Name Server already present then it identifies itself and registers with the other name server by registerNS. See image in Appendix 25

Step 4: Wait for the client or the server to connect. NS is *passive* after the initialization.

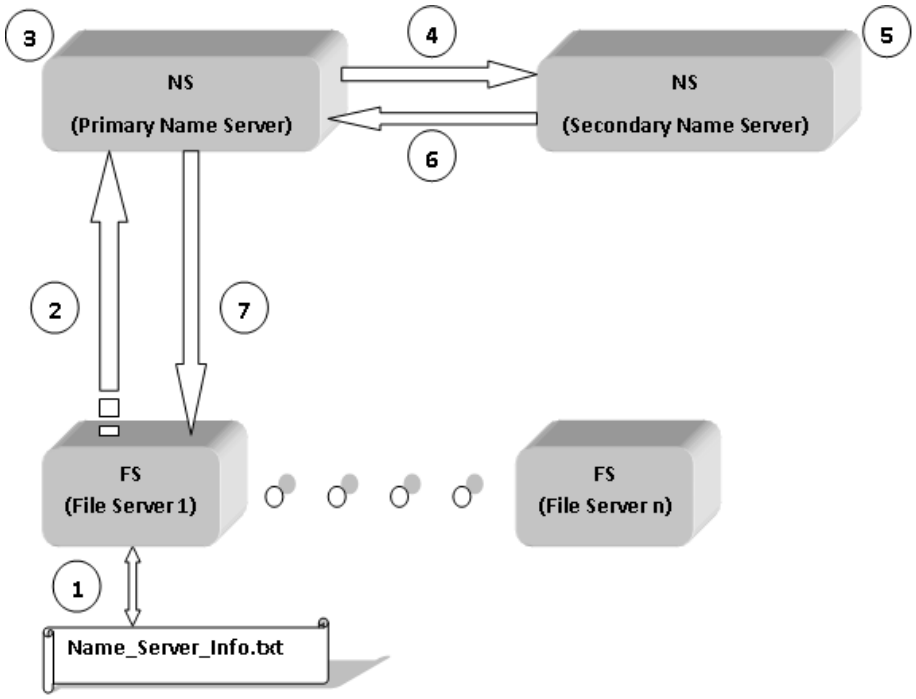

Figure 2: Registration of File Server to Name Server

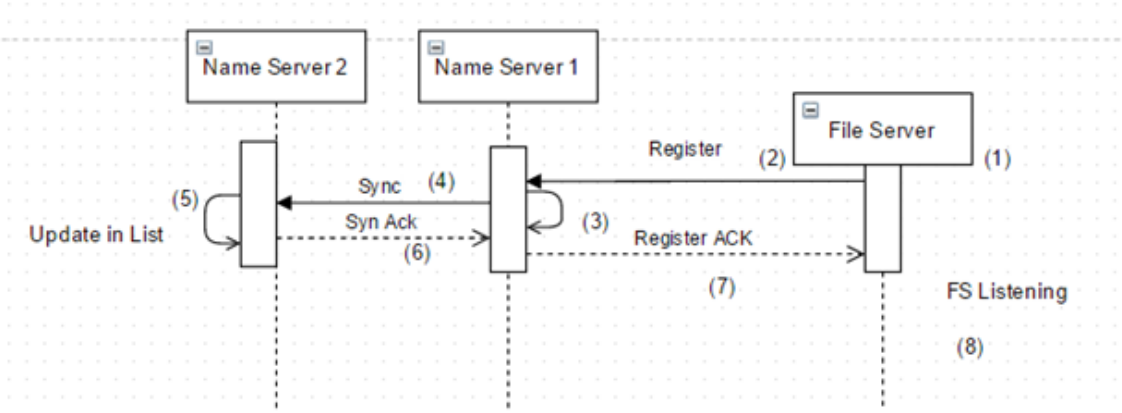

Figure 3: Registration of File Server to Name Server time diagram

3.2.2 Bootstrapping File Servers and Synchronization between Name Servers

To bootstrap the File Server(s), the software component at FS is run to initialize the machine. As we are not hard-coing IP address and Port Number of any component to improve portability, NS makes a system call to get it's IP and port number.

Step 1: The FS accesses the well known file 'Name_Server_info.txt' in AFS to read the IP address and port number of Primary NS.

Step 2: FS then requests the NS by it's IP and Port Number to establish a connection and sends it's own IP address and port number in control packet to register itself to NS.

Step 3: Upon receiving the control request from FS, NS registers the same against the logical name in it's hash file in chain of the data structure.

Step 4: Primary NS then relays the same packet or issue a register request in proxy to Secondary NS.

Step 5: Upon receiving the control request from primary NS, secondary NS registers the same against the logical name in it's hash file in chain of the data structure to keep it in sync.

Step 6: Upon successful registration, secondary NS then sends an acknowledgement to primary NS. See image 31

Step 7: Primary NS then relays the acknowledgement back to the FS assuring that it has been registered successfully to both the NS.See image in Appendix 30

Step 8: After the registration, the SFTP server is ready to listen to the connection requests from the clients. Till a connection is established by the client, FS acts as a *Listener*.See image in Appendix 29

3.2.3 Resolving Logical Naming of File Server and Interaction between Client and Name Server

To bootstrap the Client(s), the software component at Client is run to initialize the machine. As again we are not hard-coding IP address and Port Number of any component to improve portability, Client makes a system call to get it's IP and port number.

Step 1: Client accesses the well known file 'Name_Server_info.txt' in AFS to read the IP address and port numbers of both Primary and Secondary NS and maintains it in cache.

Step 2: Client then randomly chooses one of the NS and requests the NS by it's IP and Port Number to establish a connection. Client sends a resolve request to the NS to obtain IP Address and Port Number of the FS against logical name.

Step 3: Upon receiving the resolve request from client, NS checks for the linked FS in dynamic array list [3] against logical naming. Out of all listed FS, NS chooses a FS in round-robin fashion and thus the NS also acts as a *Load Balancer* for FS. NS then sends IP and Port Number of the NS to client. See image in Appendix 34

Step 4: Upon receiving the IP and Port Number, Client then tries to establishes a connection with the FS. If the connect request fails client tries for 2 more times and repeats the steps from step 2 to get new FS information from the NS.

Step 5: Server accepts the connection, forks a private port for communication and responds with the details of the private IP and port.See image in Appendix 35

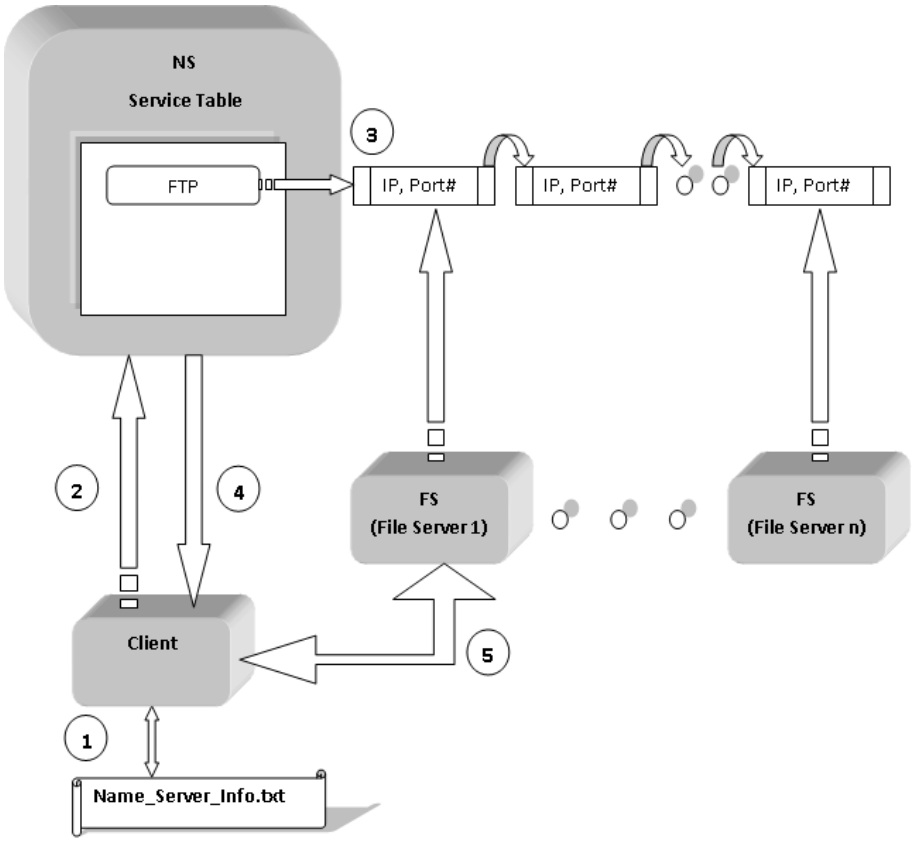

Figure 4: Resolving Query from Client by Name Server

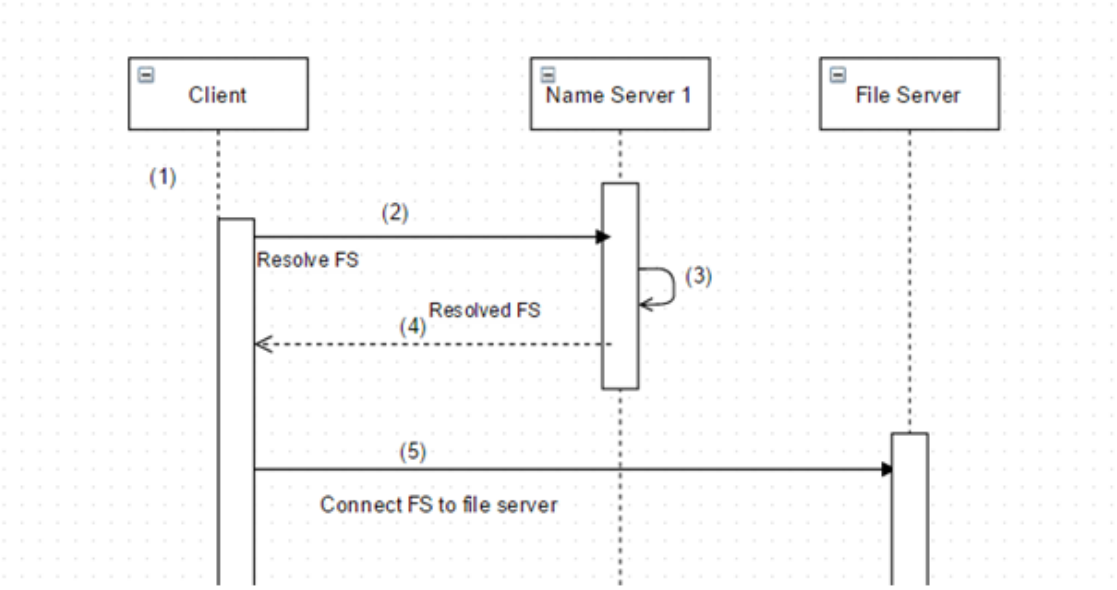

Figure 5: Resolve request from Client to Name Server

3.2.4 Interaction between Client and Server

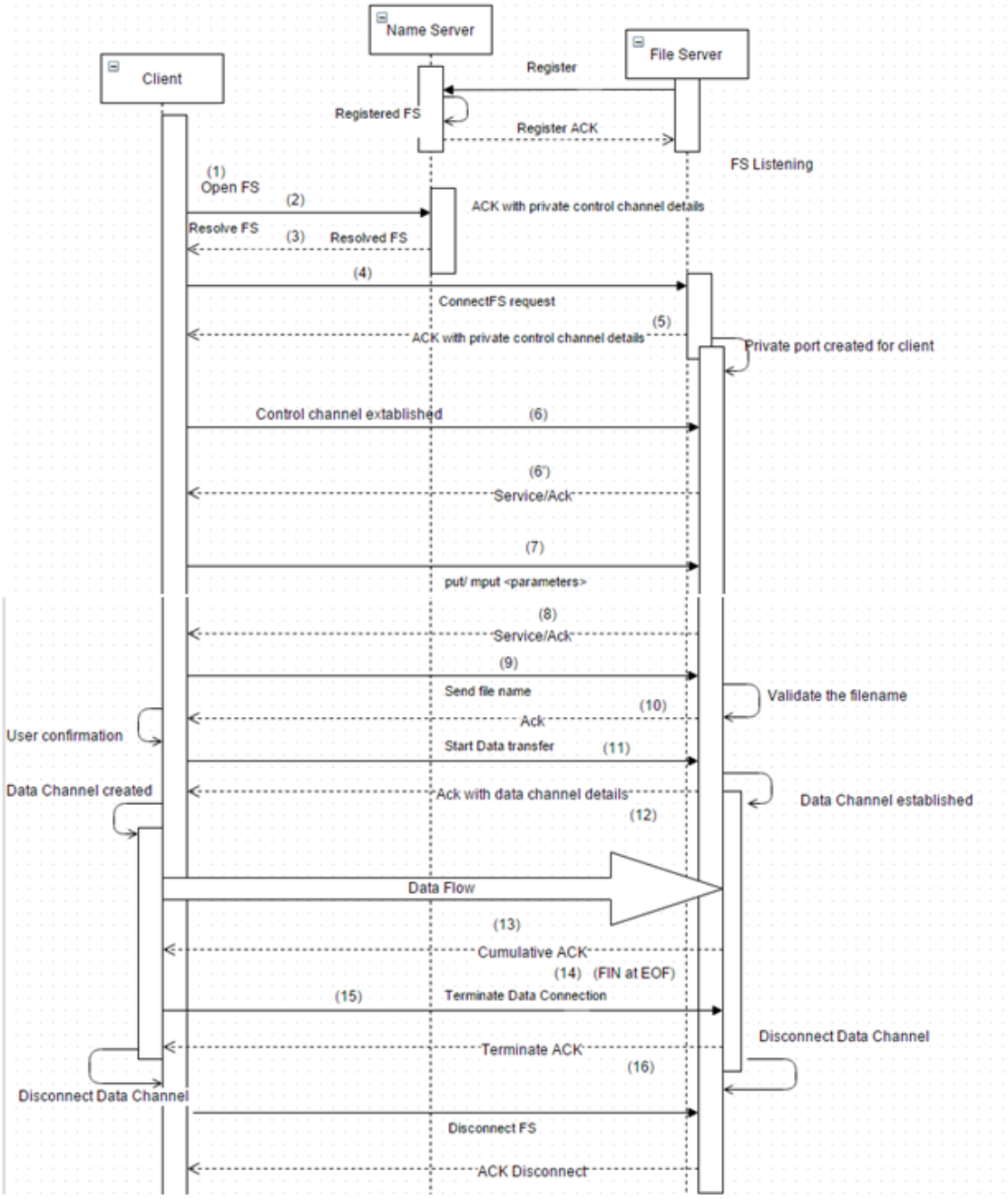

Figure 6: Interaction between Client and file server: Complete cycle

Connection and interaction between client and FS is on demand. Now that the name server and the file server are running, below are the steps that are involved in the SFTP communication between client and file server.

Step 1-5: Initial steps are the interaction between Client and NS given in previous subsection.

Step 6: Client and Server establish a private control connection to exchange commands and control signals over it. All the requests except the file transfer are handled in

the control connection like current working directory, change in current working directory and list the files in current working directory.

Step 7: When the client is ready to transfer a file, it requests a data connection with the server to share data packets. The data connection is only requested on the issue of get, mget or put, mput commands.

Step 8: FS acknowledges the request.

Step 9: On the issue of get or put command, send the details of file to file server.

Step 10: File server validates the file and acknowledges the client.

Step 11: Based on the command, the sender issues the start command for data transfer.

Step 12: Receiver creates the data port and acknowledges the sender with data port details.

Step 13: Sender creates the data port and transfers the file segment by segment over the data channel.

Step 14: Receiver sends the cumulative ACK for the received segments. On receiving the end of file, receiver issues the FIN signal.

Step 15:Sender ACKs with FIN ACK to confirm to closure.

Step 16:Receiver ACKs back and both the parties terminates the data connection gracefully.

3.3 Protocol Layering in Details

The protocol layering in Section 2 can be visualized at high level in the figure below.

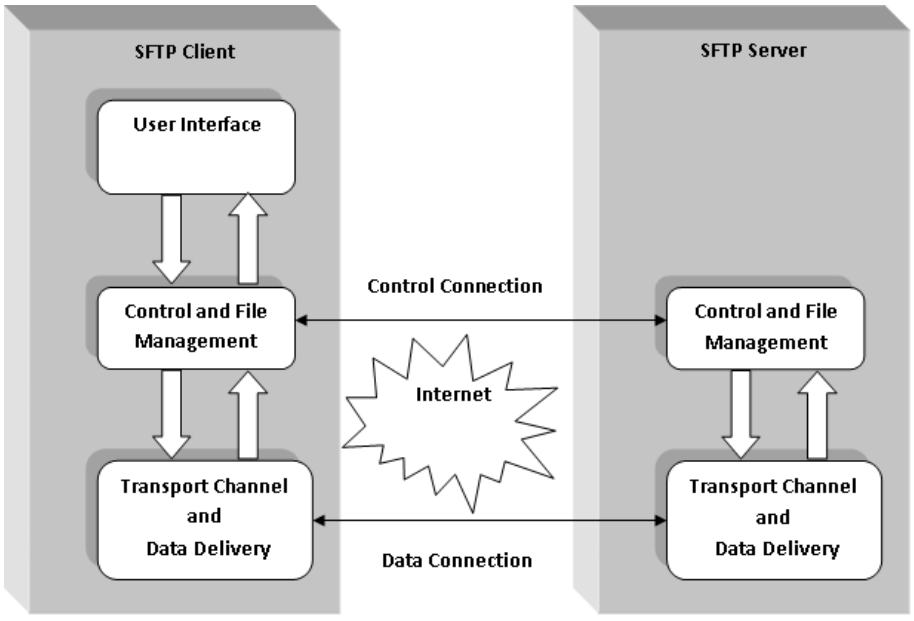

Figure 7: SFTP Layered Architecture

3.3.1 User Interface

User Interface is exposed to end users to issue commands through which they can interact with the application or rather underlying system. Based on their nature of interest or

intention, they can be classified as below.

Control Commands:

1. *SFTP*: This command is used to get into the environment; if a logical name of FS is specified with the command then it also establishes a control connection between the client and the server. See image in Appendix 36

2. *OPENFSName*: If the user doesn't provide host name as part of SFTP connection, then user specifies this command to connect to the FS. See image in Appendix 62

3. *CLOSE*: This command is used to terminate the connection between Client and the FS but the client application is still active and can connect to FS again issuing OPEN command. See image in Appendix 60

4. *ABORT*: This command cancels any immediate command being performed. **NOT IMPLEMENTED**

5. *QUIT*: This is issued to exit the SFTP environment; if the control connection is active this command closes the connection and exits the client application. See image in Appendix 65

6. ?: This command is user help, prints all the commands with brief description.

File Navigation Commands:

1. *LLS*: To list the directory contents of the local machine, this command is local and doesn't require client to establish any connection with the server.See image in Appendix 40

2. *LCD LocalDirectoryName*: To change directory at the local machine, doesn't require client to establish any connection with the server.See image in Appendix 43

3. *LPWD*: To show the present working directory at the local machine (Client), doesn't require client to establish any connection with the server. See image in Appendix 39

4. *RLS*: To list the directory contents of the remote machine (FS), the control connection between client and the server must be established before issuing the command.See image in Appendix 41

5. *RPWD*: To show the present working directory at the remote machine (FS), control connection must be established between client and server before issuing this command.See image in Appendix 39

6. *RCD RemoteDirectoryName*: To change directory at the remote machine (FS), control connection must be established between the client and server before issuing the command.See image in Appendix 42

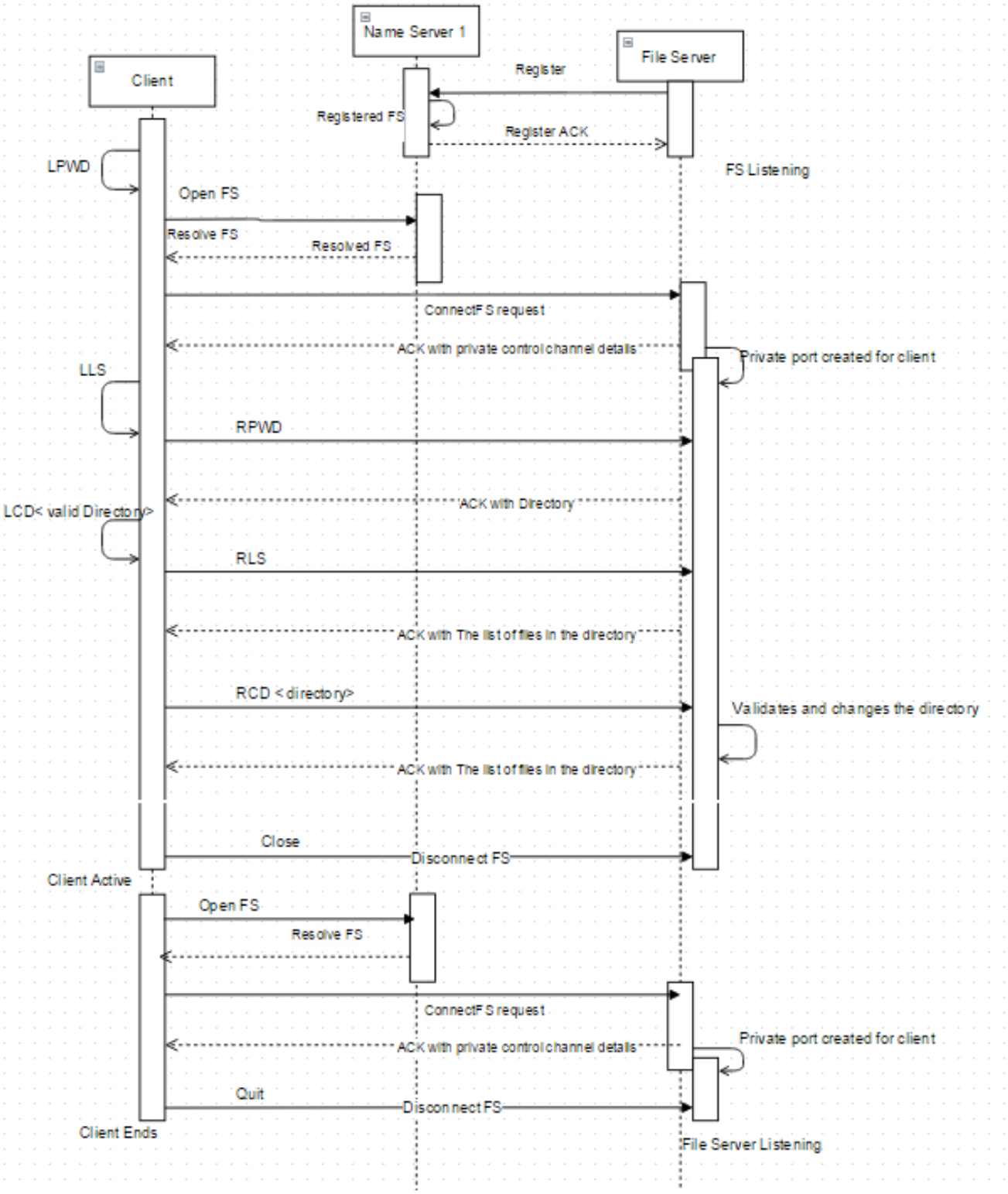

Figure 8: Timeline : Control Commands

File Transfer Commands:

1. *MGET parameter*: Request to copy multiple files from the FS to Client Local Machine. The parameter can be passed in different ways like *.* to copy all the files in current directory of the server, *.txt to copy all the text files on the current directory of the directory, D* to copy all the files beginning by the letter D on the current directory.See image in Appendix 55

2. *GET*: Request to copy the specified file from the FS to Client Local Machine. See image in Appendix 48

   Steps involved in implementation of get/mget command:

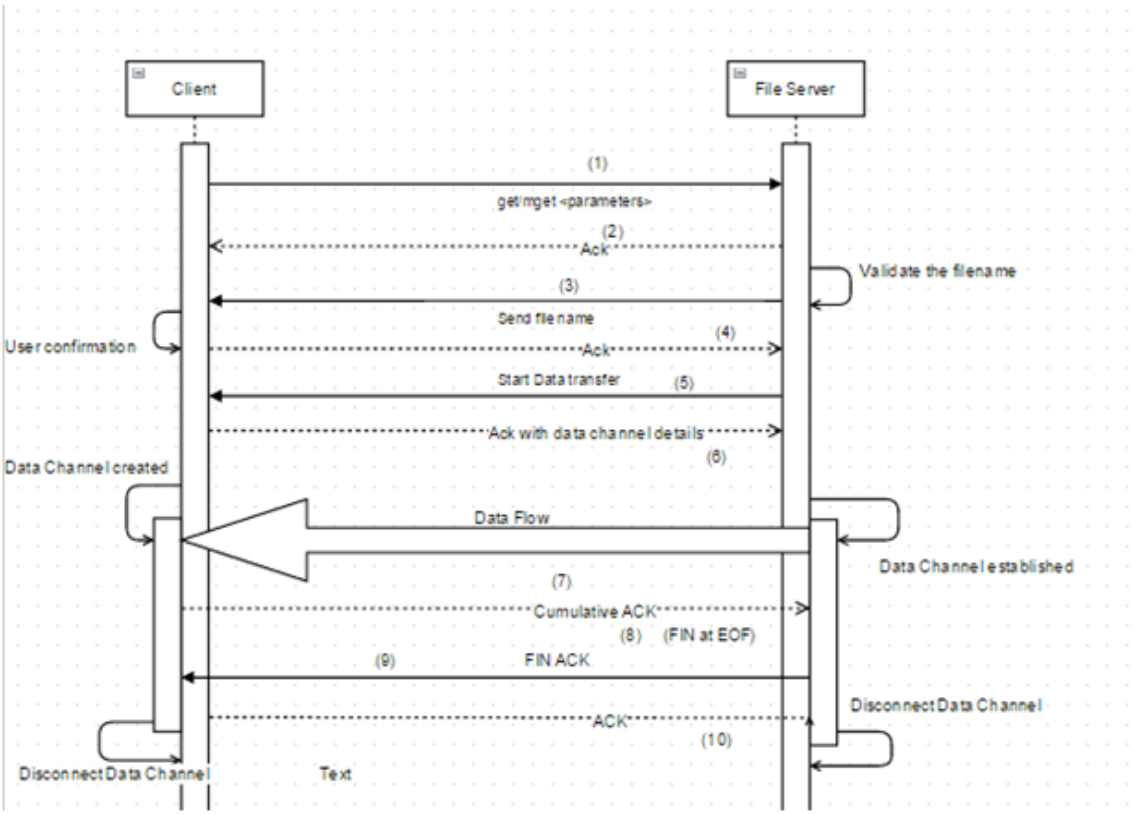

Figure 9: Timeline : GET/MGET Command

Step 1: Client issues the *get/mget* command with the filename.

Step 2: FS acknowledges the request.

Step 3: FS prepares the list of filenames and validates the file and sends the list to Client.

Step 4: Client confirms the list and ACKS with user input if there are any duplicate file which needs to be over written.

Step 5: FS issues the start signal to initiate the data transfer over data channel depending on the status of the ACK.

Step 6: Client on receiving the start, acknowledges the client with the data channel details and listens to the FS for data.

Step 7: FS starts the data transfer over data channel.

Step 8: Client receives the data and ACK's for each of the segments. On receiving the end of file segment, receiver issues the FIN signal to confirm that the file was successfully received.

Step 9: FS ACKs with FIN ACK to confirm back the closure of data channel.

Step 10: Client ACKs back and both the parties terminate the data connection gracefully.

3. *MPUT parameter*: Request to copy multiple files from Client Local Machine to the FS. The parameter can be passed in different ways like *.* to copy all the files in current directory of the server, *.txt to copy all the text files on the current directory of the directory, D* to copy all the files beginning by the letter D on the current directory. See image in Appendix 56

4. *PUT*: Request to copy the specified file from Client Local Machine to the FS.See image in Appendix 49

   Steps involved in implementation of *put/mput* command:

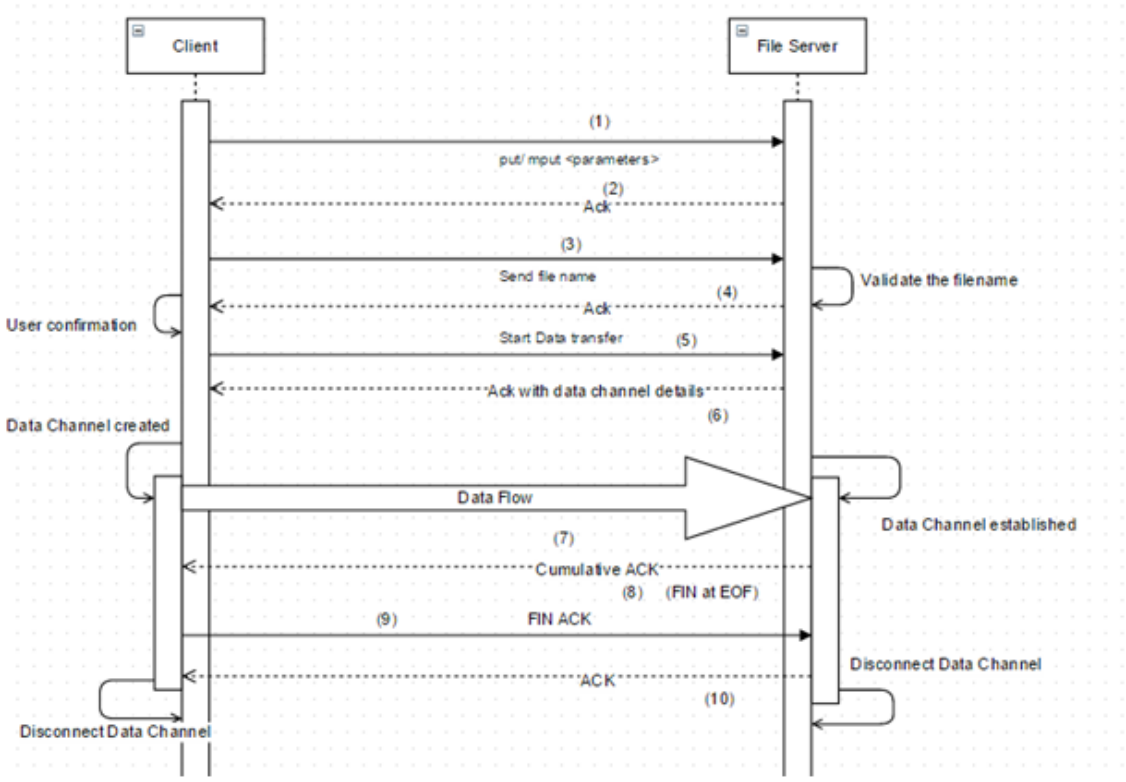

Figure 10: Timeline : PUT/MPUT Command

Step 1: Client issues the *put/mput* command with the filename.

Step 2: FS acknowledges the request.

Step 3: Client prepares the list of file names and validates the file and sends the list to Client.

Step 4: File server validates the file if it is available in remote machine and user has valid access rights on the file. After the validation FS acknowledges the client accordingly.

Step 5: Client issues the start signal to initiate the data transfer over data channel depending on the status of the ACK. If the file has to be over-written user confirms the action.

Step 6: FS on receiving the start, acknowledges the client with the data channel details and listens to the client for data

Step 7: Starts the data transfer over data channel, till the receiver receives end of file.

Step 8: FS receives the data and ACK's for each of the segments, on receiving the end of file, FS issues the FIN signal to confirm that the file was successfully received.

Step 9: Client ACKs with FIN ACK to confirm back the closure of data channel.

Step 10: FS ACKs back and both the parties terminate the data connection gracefully.

3.3.2 Control and File Management

The functionality performed by Control and File Management (CFM) Layer can be broadly classified as *Command Interpreter and Translator* and *File Management.*

Command Interpreter and Translator

CFM acts as an interface or intermediary layer between UI and TC. Thus it's major responsibility is to interpret and in turn translate the commands issued by end users at UI level and invoke function primitives from TC. It also identifies the category of commands while invoking system primitives so that each command can be treated separately and effectively based on their respective category like clear distinction between Control Commands and Data Commands.

File Management

CFM also takes care of the arguments passed to the UI commands by mapping and identifying proper file requests from the AFS and thus acts as a File Management layer. As a file manager it also allows users to navigates through the directory structure, change working directory along with read and write files under them.

As a file manager, it also needs to handle several exceptions related to file access because in a distributed system with File Servers supporting concurrent clients collision and conflict of access is a common phenomenon. The exceptions (not exhaustive) associated with file access and directory navigation can be captured as below.

1. **Bad File Request:** If the file to be requested as part of MGET, GET, MPUT or PUT from UI is not present or deleted, then it is to be identified and send File Not Found exception to the UI. In the implementation, the file server or client

depending on the direction of transfer validates the file is present or not and displays the exception. See image in Appendix 46

2. **Bad Directory Request:** Much similar to the above exception as part RCD if the asked directory is not valid, effective message is displayed to the user.See image in Appendix 42

3. **Write-Write Conflict:** While servicing multiple clients, if at any given instance of time many clients are trying to write into a single file; then exception can arise. This is achieved upon acquiring exclusive write lock by system call for the first client being serviced and the second user can be notified the same. See image in Appendix 75 ,76

4. **Write-Read Conflict:** If an user requests to read a file when some other user is writing into it, this situation may arise. Though this has nothing like conflict for write lock, but the user who is trying to read will not get most updated version of the file.

On encountering any of these exceptions, file manager should handle it effectively by requesting TC to send effective control signals to the other party and at the same time CFM of other party should effectively analyze the control information received and send user friendly verbiage back to UI layer to display to the clients.

3.3.3 Transport Channel and Data Delivery

This layer performs functions like below.

Processing Files:

The first responsibility of this layer is to process the file sent by CFM to handle different aspects like as followings.

1. Big endian or the little endian system: both systems uses different method to store data in memory, standard byte order for networks is big endian. To handle different machine systems, we have to check for the endianness of the machine. Before transferring data from such machine to the network, data has to be converted to big endian order and while transferring the data back to the machine we have transform again back to little endian system. This is implemented using the functions *htonl*() and *ntohl*().

2. Identifying the file type: Files can be either ASCII or Binary type. If we deal them separately then a situation may arise where client is encoded to serve ASCII and FS to serve in Binary mode or vice versa. To avoid the confusion, irrespective of the nature files are going to be read as bit stream to construct data packets.

Connections:

One of the major responsibility of this layer is to establish connections between different components. By analysing the nature of requires we can easily identify two types of connections.

1. Control Connection:

   This connection is long lived and established between any OPEN command and CLOSE command. This channel is used to transmit control information like commands, Request, ACK, NACK, and other signals to effectively manage the system. Hence the packet structure involves message type and message mode to define the purpose of the packet.

   The information carried by this connection is very small in size and thus reliability is not a major factor here. Moreover this is a long lived connection, so best design choice to implement this is by using UDP.

   Every commands or control packets are assigned with an unique identifier to differentiate between each other.

   The structure of packets used by this connection look like below, and the brief description is followed by the figure:

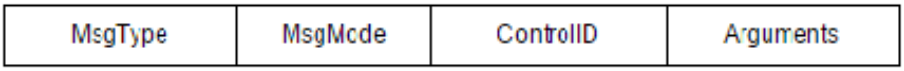

Figure 11: Structure of Control Packet for UDP

   (a) Message Type: Distinguishes the type of control packet, and the purpose of it. These are the message types we have defined for the implementation of control channel:

      i. Register FS (0000): To register FS with the name server.
      ii. Resolve FS (0001) : To resolve FS requested by Client to NS.
      iii. Register NS (0010): Identification of name servers with each other.
      iv. Sync NS (0011): Sync signal between the name servers to update each others dynamic array list [3].
      v. SFTP Command (0100): Indicates that the packet is a SFTP command and this is further defined by the control ID.
      vi. Connect FS (0101): Client request for connection with the FS.
      vii. Disconnect FS (0110): Closing the connection with FS.
      viii. No Control Id (0111): Indicates that Control ID field is empty for this message type.
      ix. Establish Data Connection (1000): For data connection establishment.
      x. Terminate Data Connection (1001): Tearing down the data connection.
      xi. Send File (1010): To indicate that the control packet contains file name to be validated.

xii. Start Transfer (1011): Start of the data transfer.

(b) Message Mode: This defines the mode of the packet, if it is the request or the response to a request.

i. Request (00): Indicates that the packet carries a request for the mentioned message type.

ii. ACK (01): Indicates that the packet is an Acknowledgement to mentioned message type.

iii. NACK (10): Indicates that the packet is a NACK for negative acknowledgement for the message type mentioned.

(c) Control ID: It determines the command required to interact with the remote machine. Control ID carries the code only when the message type is SFTP command so that the CFM translates the command to native functions accordingly. The commands are as mentioned below:

i. RLS (000): Indicates that the command is rls.

ii. RCD (001): Indicates that the command is rcd.

iii. RPWD (010): Indicates that the command is rpwd.

iv. GET (011): Indicates that the command is get.

v. PUT (100): Indicates that the command is put.

vi. MGET (101: Indicates that the command is get.

vii. MPUT (110): Indicates that the command is mget.

(d) Argument carries the parameter for each command or system requests.

2. Data Connection:

This connection is short lived and established during any file share command. This channel is used to transmit actual data packets of files on exchange.

The information carried by this connection is effectively very large in size and thus reliability is a major concern here. Moreover this is a short lived connection, so resources are kept dedicated for utilization to ensure reliability of delivery. We have implemented the data connection using UDP with few additional functionality Error and flow control using Go back N and CRC [1]protocols to provide the required reliability over the transport channel. We have assumed in our implementation that the maximum data length held by the packet is the MTU defined in our configuration file and total packet length would be MTU + over head of the packet.

The structure of data packet used by the data connection looks as below:

Data Identifier indicates if the packet is data or the final handshake message:

(a) If set to DATA (0) the packet defines the data and follows by the below representation.

i. Sequence number: Representing the sequence number of the packet

ii. Fragmentation flag: Fragments available or not

A. If set to 1 indicates there are more packets of data to come.

B. if set to 0 indicates the last packet or end of file.

iii. Data Length: Corresponds to the length of the data attached to the packet.

iv. CRC: CRC byte is added to the packet by calculating the CRC [1] of the data segment.

v. Data field: Represents the actual data segment in the packet.

(b) If set to FIN message (1) follows by the FIN message type which represents final hand shake signal exchanged between the sender and receiver:

i. Fin (00): FIN is sent when the receiver receives the end of file.

ii. finAck (01): FIN ACK is sent by the sender on receiving the FIN and agrees to the channel closure request.

iii. finalAck (10): Sent by the receiver as the final message and both the parties terminates the data connection gracefully.

The structure of the ACK packet will have only the sequence number, which represents the next packet to be sent.

Figure 12: Structure of Data packet for UDP: DATA mode

Figure 13: Structure of Data packet for UDP: FINMSG mode

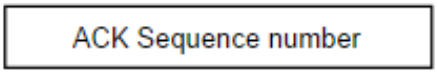

Figure 14: Structure of Data Acknowledgement for UDP

As part of Data Connection, system should provide reliable communication over unreliable network. So, below concerns are to be addressed by the SFTP.

1. **Error Detection:**

   We implement the CRC [1] to detect the error in the data segment. For each d bits of data we append r bits of CRC to form a frame of d+r bits. Before that the sender and the receiver has to agree on a CRC generator G of r+1 bits to calculate r bits to append to each d bits of the data frame. The receiver on receiving the

frame divides with the key G , if there is a reminder the receiver knows that an error has occurred in the received data sends an ARQ to the sender. CRC can detect a burst error of less than r+1 bits. The ARQ technique used is discussed in flow and congestion control as error detection and congestion control go hand in hand.

2. **Flow and Congestion Control:** The purpose of the flow and congestion control is to regulate the traffic in order to prevent the overflowing of the buffers at the receiver and resulting in packet drops. We have a number of techniques like Stop and Wait, or Sliding Window protocol to use from.

   (a) **Stop and Wait protocol:** In this protocol the sender transmits a segment, and waits for the receiver to send an acknowledgement. The sender does not transmit the next segment until and unless the acknowledgement (Ack) is received from the receiver. This method is very effective for short and fast links where the propagation delay is very small. But if we consider slower and longer links the bandwidth of the link is wasted in waiting for the Ack.

   (b) **Sliding Window Protocol:** In this protocol the sender and receiver agrees on the window size based on the receiver buffer and sender capacity to send the frames. The sender is allowed to send multiple frames together based on the window size and waits for the Ack for the frames sent. Each frame sent is numbered to track the status of the frame and Acks are usually numbered the next frame to receive. We can use 2 techniques to prevent the errors in sliding window protocol:

      i. **Go back N protocol:** The receiver asks the sender to retransmit all the frames from the moment there is an out of sequence frame received. Go back N protocol is used when the maintaining the buffer space is a concern.

      ii. **Selective Repeat:** The receiver buffers all the received frames and only asks the sender to resend the missing or corrupted frames. To achieve this we need to maintain buffers as well as linked list to maintain all the frames received in the previous window to check for duplicates.

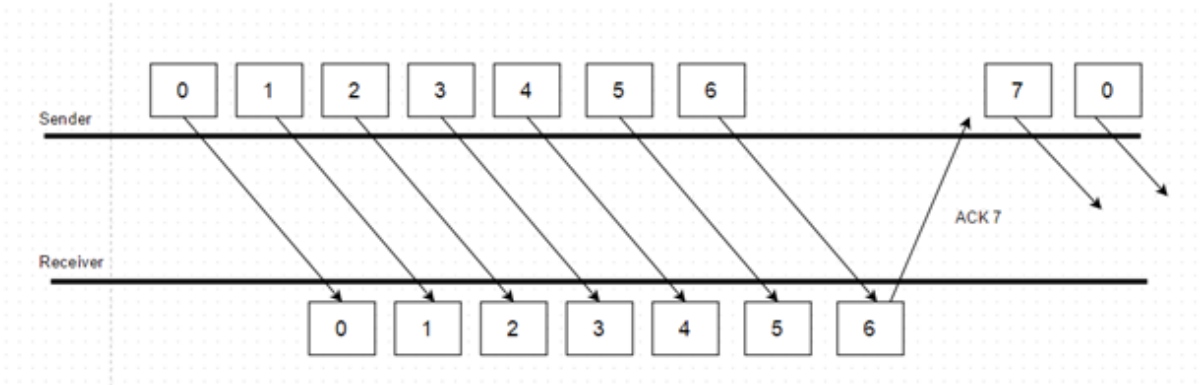

Figure 15: Go back N with cumulative ACK for successful transmission

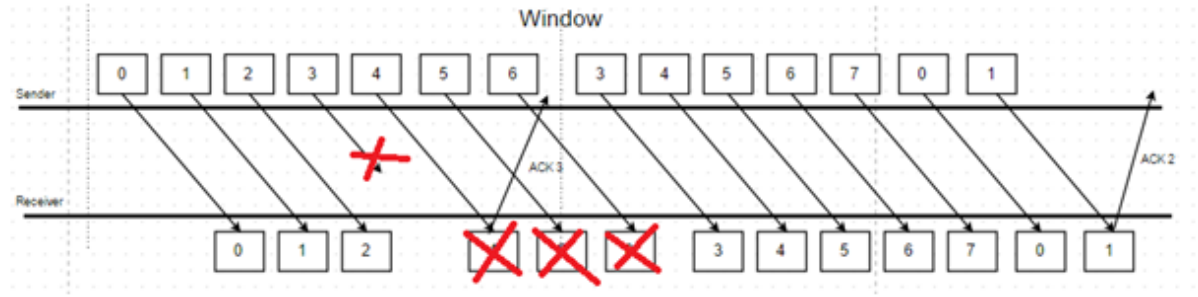

Figure 16: Corrupted or lost Packet

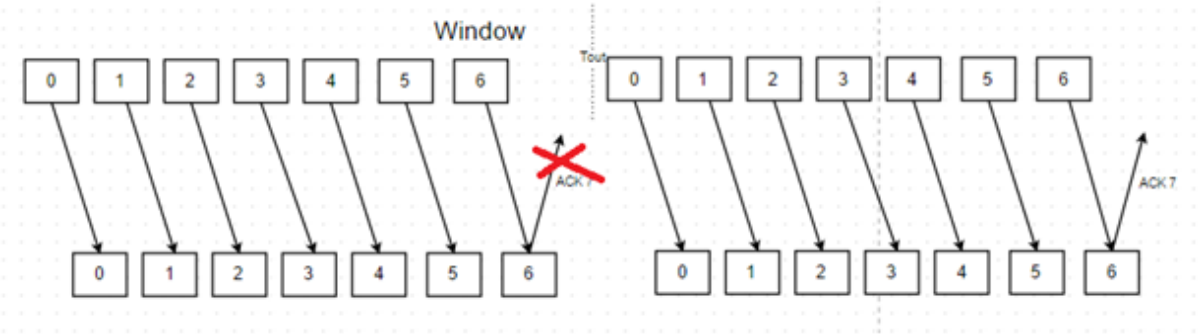

Figure 17: Lost ACK

We prefer to implement the Go-back-N protocol as it needs less resource in terms of data structure when compared to Selective Repeat.

Factors to be considered to Implement Go Back N protocol:

1. Decide on the window size between the sender and receiver based on the receiver buffer and sender capacity. If k is the number bits in the sequence, the maximum size of the window is $(2^k) - 1$.

2. Decide on the timeout between the sender and receiver based on the worst case propagation and processing delay between the sender and receiver.

3. Sender transmits multiple frames in a sequence, a maximum of window size S_w at a time, and starts the timer for last frame sent.

4. Receiver send the ACK (n+1) for the nth frame received in sequence.

5. Sender resets the nth timer after each Ack (n+1) received.

6. In case of any frame received in out of order sequence, the receiver discards all the out of sequence frames and transmit NACK(n) (negative ACK) for the frame which was not received explicitly.

7. The sender re-sends from frame number n onwards after receiving the NACK, and the sequence number wraps around and continues as shown in the figure above.

We implemented the sliding window protocol with configurable window size, so that we can set the window size to 1 and check the performance of stop and wait protocol

against the sliding window protocol. We are using the cumulative ACK for the GO-back-N protocol as explained above.

4 Configurable Items

A configurable item removes the dependency of hard-coding in the implementation by decoupling and placing them somewhere accessible by the implementation globally. The placement of a configurable item can be any of the two given options - *header file* and *external configurable file. In our implementation, we placed these configurable items, to be used by the system, in an external file and the header file of the implementation will have pointer to the location of the header file.*

Followings are a few identified (not exhaustive) configurable items to be used by the system upon realization.

MTU: *Maximum Transmission Unit*
Window_Size: *Window Size*
P_Err: *Probability of packet in error*
P_Cong: *Percentage Congestion*
Name_Server_Info: *Fully qualified path for Name Server file*
SFTP_FS: *Logical name of the file server*
Attempt_No: Number of try
Time_Out: Time out period
Inj_Error_Test: *Error injection flag*
Stat_Analysis_Log_File: *Fully qualified path for Statistical Analysis Log file*

5 Requirements for Implementation

This section captures the requirements for the implementation of the system and realization of the protocol in terms of platform, tools etc.

Platform:

The application is designed and implemented to be used in ***UNIX*** or ***LINUX*** platform.

Language for Implementation:

Implemented the project using ***C/C++***.

Tools:

Version Management Tool:

Git- to manage sharing of source code, collaboration within team, version management etc.

Documentation Tool:

LATEX- to create documents like design document and final report.

Statistical Analysis Tool:

MATLAB- to formulate statistical approaches, formulate mathematical models and plot curves for conclusion.

Build Tool:

Make- to build the software bundle automatically and accommodate changes on the fly.

6 Test Scenarios and Emulation

As part of test strategy we test each component individually and independently and also performed integration testing. Tests will be performed on a set of files (large enough and in large in number) for a longer period of time. Files to be shared between clients and servers are classified based on test scenarios and maintained under designated folder structure. Test scripts will access this folder structure to issue connection between components. Scripts are to be automated to run in loop to fetch files, establish connection and log performance metrics.

6.1 Configuration Items validation testing

Before boot strapping of each of the module, all the configuration items in the configuration file is validated and populated. If any of the configuration item is invalid boot strapping fails with appropriate message for valid configurable item.

1. Check if the $Time\_Out$ is valid and greater than 0 always.
2. Check if the number of attempts is numerical and positive.
3. Check the name server info file is accessible over the path, and is readable and writable.
4. Logical name for the file server is updated with valid string.
5. Window size should be numerical and always greater than 1.
6. MTU is always numerical and positive in the range of 1 and 1500 bytes.
7. Test error injection flag is either 0 or 1 indicating if error injection is done or not.
8. P_Err is defined with any decimal value in the range 0 and 1.
9. P_C is defined with any decimal value in the range 0 to 100.

10. Check the statistical analysis log file is accessible over the path, and is readable and writable.

Tested OK.

6.2 Module Testing:

Test the individual modules, and check if the behaves as expected.

1. Name server boot strapping: Single name server is boot- strapped, gets the IP from the system. Creates a UDP control socket to listen. Reads the well-known file $Name\_server\_info.txt$ and update its $nsID$, IP and port number into the well-known file.
   Now the name server is ready to listen to requests from File servers and clients. See image in Appendix 25

2. Booting the second Name Server: Boot strapping another name server.See image in Appendix 26

   (a) The name server gets the IP from the system and creates a control UDP socket as process.

   (b) Now reads the well-known file $Name\_server\_info.txt$, Updates it $nsID$, IP and port number into the $Name\_Server\_info.txt$.

   (c) Identifies the other name server being available and register itself to the other name server and also keeps a copy of the details with itself to sync and update the file server list between each name server.

   (d) Name server 1 registers with the name server2. See image in Appendix 27

   (e) Ready to listen to the requests from file servers and clients.

3. File Server boot strapping:

   (a) File Server is able to get its IP and create a UDP control socket to listen.

   (b) Reads the $Name\_Server\_info.txt$, reads both the name server info and picks one of the name servers randomly.

   (c) Send its IP and port number to the name server 1 successfully in this case.

   (d) Name server 1 accepts the File server IP and port and registers into its dynamic list [3].

   (e) Name server 1 requests to Name Server 2 for registering the file server info in its dynamic [3] list. See image in Appendix 31

   (f) Name server 2 successfully registers and ACKs the Name server 1. See image in Appendix 30

   (g) Name server 1 confirms the registration with the file server. See image in Appendix 29

4. Starting up the Client:
   (a) Run the Client application using the command *SFTPClient* or by using *SFTPClient SFTP_FS* (logical name of the server to connect). See image in Appendix 32
   (b) Use Open command to connect to the file server, on issuing the command OPEN *SFTP_FS*. See image in Appendix 62
      i. Client reads the name server info file, picks one of the name servers.
      ii. Requests the name server to resolve the file server logical name into meaningful IP and port.
      iii. Name server checks the dynamic list [3] of name servers and picks the registered file server details and responds if present.
      iv. On receiving the details, client requests the file server to connect.
      v. File accepts the request of the client and forks a private control channel to provide service to the client and keeps the known port to listen to next client. See image in Appendix 35
5. Running the second Client: The same procedure follows, client requests name server to resolve the file server logical name with known IP and port number. The file server listens to the request of the new client on the known IP and port, forks a private port to service the second client and the FS again starts to listen to the new client over the known port. See image in Appendix 36

6.3 Integration testing:

All the components are run and tested for all possible scenarios for each command:

We begin with 2 NS, 1 FS, and 1 Client:

Check each of the commands, if we get the desired outcome:

1. LPWD: Show the current directory in the local machine, does not request to server as it is local.
2. RPWD: Show the current directory in the remote machine, gets the detail from the file server. See image in Appendix 39
3. LLS: List the files in current directory in local machine. See image in Appendix 40
4. RLS: List the files in current directory in remote machine. See image in Appendix 41
5. RCD <parameter>: Change the directory in remote machine, if directory is present go to the mentioned directory else pop the error message that the file directory is invalid. See image in Appendix 42
6. LCD <parameter> : Change the directory in local machine, and also should be able to catch the invalid directory. See image in Appendix 43

7. GET <file name>: copy the file from remote to the local machine at the current working directory if the file is present in the server. Ask the user if the file needs to be over written. See image in Appendix 48

   (a) Client checks the file name if it is available at local machine, if available asks the user if he wants to over write.

   (b) If yes provides the file name and details with server over control connection. Server validates the file name and access rights.

   (c) After validation server opens the data socket, provides the detail of the data connection to client.

   (d) Client acknowledges the details with its data socket details.

   (e) Data transfer begins on the data channel.

   (f) After the completion of the data transfer, the data channel is terminated.

   Case 1: File exists in FS but not in Client: Successful transfer. Tested Ok

   Case 2: File exists in both FS and Client: Application asks the user if he wants to over write the file, based on the user feedback the file is transferred or not transferred.
   If no, there is no data connection created between the server and client. Tested Ok.See image in Appendix 44 If yes, the file is over written: Successful transfer. Tested Ok. See image in Appendix 45

   Case 3: Invalid file / File not found in FS: Displays the appropriate error message. Tested Ok. See image in Appendix 46

8. PUT <file name>: Copy the file from local to the remote machine at the current working directory, verify the list of files, if the file needs to be over written check the write access and confirm with user to overwrite.

   (a) Client sends the file name he needs to copy into the server machine over control channel.

   (b) Server validates the file if it is available at remote machine, if available checks the write permission.

   (c) After validation if the file has write permission asks the user if he wants to over write otherwise denies.

   (d) If conditions satisfy, server provides the data connection details to client.

   (e) Client acknowledges, and establishes a data connection.

   (f) Data transfer begins on the data channel.

   (g) After the completion of the data transfer, the data channel is terminated.

   Case 1: File doesn't exist in FS, exists in Client: Tested OK.

   Case 2: File exists in both FS and Client:Tested Ok . See image in Appendix 49

   Case 3: File doesn't exist in Client: Tested Ok.

   Case 4: If the file exist in FS and the user does not have write access: Application does not allow the operation, Tested OK. See image in Appendix 51

9. MGET <parameter>: Copy the list of files from remote to the local machine at the current directory, ask the user if any of the file needs to be overwritten. The process followed by the mget is same as get, only that it takes multiple file names at a time.
   We have implemented different formats of parameters which can be given for both *mget* and *mput* like:

   (a) $mget*.*$: All the files in the current directory in remote system is selected for transfer. See image in Appendix 52

   (b) $mget*.txt$:All the text files in the current directory in remote system is selected for transfer. See image in Appendix 54

   (c) $mgetfilename1, filename2, filename3$: mentioned files in current directory in remote system are marked for transfer, if the files are already present over write warning is displayed just like in get command. See image in Appendix 55

   (d) $mgetT*.*$: All the files starting from 'T' in the current directory in remote system is selected for transfer. Similarly many combinations can be made. See image in Appendix 53

10. MPUT <parameter>: Copy the list of files from local to the remote machine at the current directory, verify the list of files, if the file needs to be over written check the write access and confirm with user to overwrite. The process followed by the *mput* is same as *put*, only that it takes multiple file names at a time. Rest of the functionality remains the same.

    (a) $mput*.*$: All the files in the current directory in local system is selected for transfer. See image in Appendix 58

    (b) $mput*.txt$:All the text files in the current directory in local system is selected for transfer.

    (c) $mputfilename1, filename2, filename3$: mentioned files in current directory in local system are marked for transfer, if the files are already present over write warning is displayed just like in put command. See image in Appendix 56

    (d) $mputT*.*$: All the files starting from 'T' in the current directory in local system is selected for transfer. Similarly many combinations can be made. See image in Appendix 57

11. Check the CLOSE command is working; it should close the control connection and not the application. SFTP prompt should be active.

    Case 1: Check with Open command again if we connect to the file server again: See image in Appendix 64

    Case 2:Check if after close, we can access FS, It should not be allowed. Tested OK See image in Appendix 64

    Case 3: Check if multiple open can be given when the client is already connected to FS. It should not allow. Tested OK

12. Check the QUIT command is working, it should close the control connection if open, close the application and control should come out of the SFTP prompt. See image in Appendix 66,65

6.3.1 Name Server Robustness Testing

1. Boot-strap multiple file servers and check if the register and sync operation works in the name server correctly. See image in Appendix 31

   Outcome: We can observe that the name server picked by the file servers are random, hence the load is fairly distributed between NS1 and NS2.

2. Run multiple clients, and check if the name servers are able to resolve all the requests and how is the load balancing handled by the name server.
   Let us run 6 clients and see how the name server resolves the file servers, currently we have 3 file servers registered with the name servers.

   Outcome: For each of the file server the load of client has been equally distributed by the name server. In our example each file server handles 2 of the clients.

3. Robustness of the Name Server: Make a name server down, and check if the other name server can handle the load efficiently.

   Step 1: NS1 is brought down.

   Step 2: Check if NS2 can handle all requests, run more clients and see if it can distribute the load equally among the 3 FS. NS2 is able to handle all the requests and service both the clients and servers.

   Step 3: Run more file servers, and check if the register requests are handled by the NS2.

   Step 4: Make NS2 down, system goes down. Client cannot resolve the file server and hence no connection.

6.4 Check the robustness of the File server:

Check the behaviour of client when the file server is down, if it can pick the next available file server.

Step 1: Boot strap multiple file servers.
Step 2: FS1 is brought down.
Step 3: Run multiple client application, client randomly picks the NS.
Step 4: NS resolves the FS and responds to client.
Step 5: One of the clients gets FS1, tries to connect to FS1 thrice and fails.
Step 6: Client repeats the above steps to get a running file server.

Tested OK.

6.5 Stress and load testing on the file server

Step 1: Check if the server can handle 10 clients at a time.
Step 2: Try changing the remote working directory to different directories for each client connected.
Step 3: Try performing the operations and validate the result if each client is independently handled.
Step 4: Now try multiple clients to point to a single directory, and perform file operations and check the behavior of the server if the read and writes are handled correctly.
For 1 NS, 1FS and 2 Clients:

Case 1: : Tried changing the remote directory, both the clients were handled independently.
Case 2: Let both the client have the remote directory, try reading the same file:
Case 3: Now try putting the same file into the remote directory, check how it behaves:
Client was denied the access as Client 2 had already started the transfer of file. Hence file concurrency has been handled. See image in Appendix 75,76

6.6 Multiple file types to be transferred of varied size

Case 1: Try transferring txt files, pdf files, mp4, xls, doc files etc and see if the transfer succeeds and the files transferred is in valid format and can be opened.
Outcome: Tested with all types of files, all files are in valid format and can be opened.
Case 2: Try transferring different size of data, less than MTU, more than MTU, multiples of MTU:

Output 1:Less than MTU, file transferred without any problem.
Output 2:Multiples of MTU, successfully transferred.
Output 3:Very large files, Tried transferring 200 MB file, successfully transferred with correct format and read correctly.
Output 4:For MTU value of 1500, file transfer fails for files of size greater than or equal to 1500 bytes.

7 Statistical Analysis

We have implemented both bit error injection [2] and percentage of congestion[2] introduced forcefully to check the behaviour of the SFTP system in a error prone and congested channel.

7.1 Delay Perfromance vs Bit Error probability,P_{Err}:

Strategy: Make a set of 10 folders each containing different file sizes. Run the client-server $SFTP$ application for each folder for $P_Err = 0$ note the total time taken against the file size. Continue the experiment for $P_{Err} = 0.1, P_{Err} = 0.2, P_{Err} = 0.3$,and $P_{Err} = 0.7$ and note down the readings in $Stat\_Analysis.log$

Plot the graph of total time taken for file transfer against file size for each of the P_{Err} using MATLAB.
Constraint: S_W is fixed, MTU is fixed, T_{Out} is fixed. P_c is fixed.
Variables: File size, P_{Err} and total time take for file transfer.
Goal: Calculate the confidence interval of total time taken for file transfer through which 95% of the sample falls.

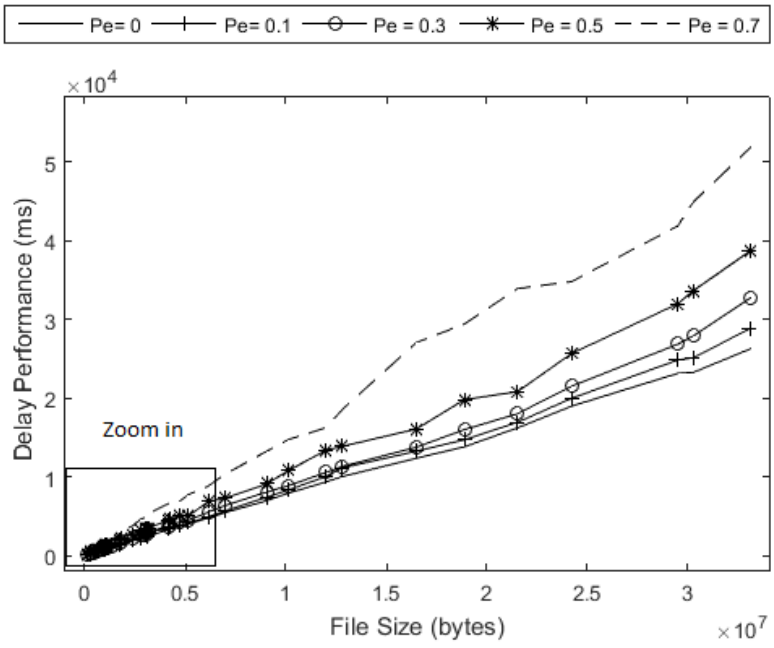

Figure 18: Go back N performance against Probability of Error

File Size(MB)	1	10	22
Mean(ms)	772	8720	17434
STD	47.6	628.9	742.3
Confidence Interval(95%)	31	410.3	485.3

Table 1: GBN: Error and congestion free channel

Analysis and Conclusion: As expected as the error probability in the channel increases the delay response of the system increases hence the performance decreases, the confidence window for different file sizes and for different error probabilities are listed as in the table. The confidence window length increases gradually with the increase in probability. We also notice that for small files which are in kilo bytes are highly inconsistent due to the queueing delays, if we zoom in the graph we can notice sharp highs and lows which indicates the inconsistent behaviour of small files due to the queueing delays occurred but as the file size increases the effects of queueing delay is not comparable to the delay response.

File Size(MB)	1	10	22
Mean(ms)	851	8967	18192
STD	49.7	810.2	900.1
Confidence Interval(95%)	32.4	561.4	623

Table 2: GBN: Probability of Error = 0.1

File Size(MB)	1	10	22
Mean(ms)	1250	11218	22630
STD	119.9	965.5	1540.1
Confidence Interval(95%)	78.3	772.5	1067

Table 3: GBN: Probability of Error = 0.5

7.2 Delay Perfromance vs Congestion level, P_c:

Strategy: Make a set of 10 folders each containing different file sizes. Run the client-server SFTP application for each folder for percentage of congestion, $P_C = 0$ note the total time taken for file transfer against the file size in *Stat_Analysis.log*. Continue the experiment for $P_C = 10$ and $P_C = 40$.

Plot the graph of total time taken for file transfer against file size for each of the P_C using MATLAB.

Constraint: S_w is fixed, MTU is fixed, $T_{Out}$is fixed. P_{Err} is fixed.

Variables: File size, P_C and total time taken for file transfer.

Goal: Calculate the confidence interval of total time take for file transfer through which 95% of the sample falls.

Conclusion: As expected the performance of the Go back N decreases with the increase in congestion, at 40$ the delay performance raises sharply when compared 0 and 10% congestion.

File Size(MB)	1	10	22
Mean(ms)	2262	21612	46367
STD	132.5	895.6	973.8
Confidence Interval(95%)	116	716.6	721.4

Table 4: GBN: Congestion Level = 10

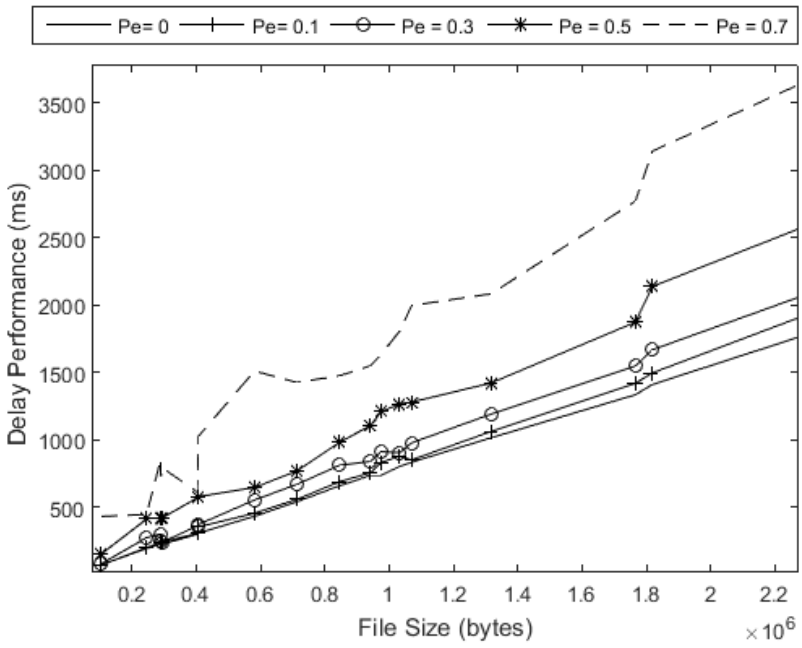

Figure 19: Go back N performance against P_{Err}

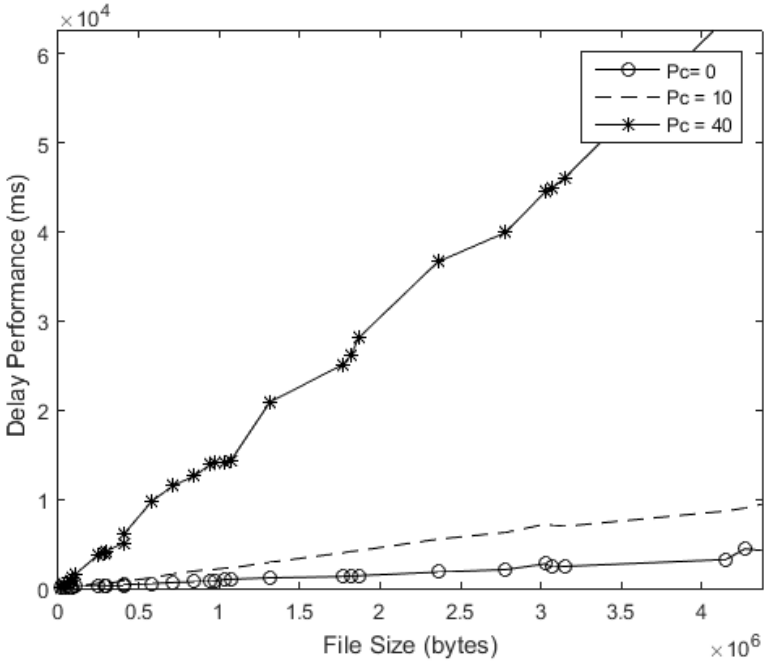

Figure 20: Go back N performance against Percentage of Congestion

File Size(MB)	1	10	22
Mean(ms)	15390	152109	322060
STD	865.4	2904.3	10258
Confidence Interval(95%)	599.7	2012.6	7108.3

Table 5: GBN: Congestion Level = 40

7.3 Performance comparison between Go back N and Stop and wait vs P_c

Stop and Wait is a sliding window protocol of window size 1, here we compare the stop and wait protocol against the Sliding window Go-back-N ARQ.

Strategy: First compare the performance of Go back N with window size of 8 and Stop and Wait protocol how they perform in data transfer. Collect the response time for transfer for varying file size and plot the graph.

Constraint:P_{Err} is fixed to 0, MTU is fixed to 1000, T_{Out} is fixed 0.5s, P_c is fixed to 0.

Variables:S_w, file size and total time taken for file transfer.

Goal: To find the performance of Go back N and stop and wait in an error and congestion free channel.

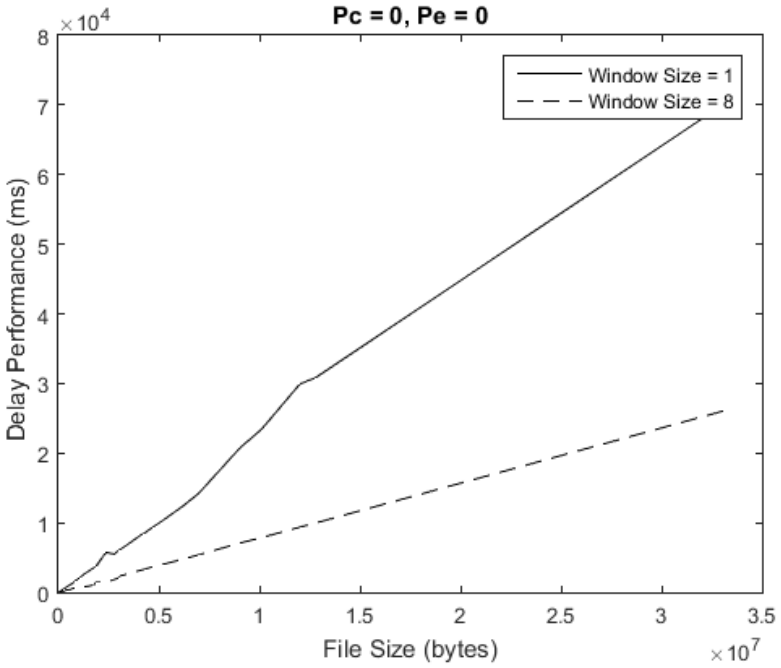

Figure 21: Go back N vs Stop and Wait

Conclusion: Based upon the graph, we can clearly conclude that the performance of Go back N is better than the Stop and wait protocol in an error free channel.

Building on the previous experiment, here we keep the file size constant and vary the congestion percentage and see how both the protocol behave.

For a fixed size of file (1MB) we vary percentage of congestion (P_c) and see how Go back N and stop and wait protocols performs. Continue the experiment for P_c=0, P_c=10, P_c=20, $P_c = 30$, $P_c = 40$, $P_c = 50$ and $P_c = 60$ for window size $S_w = 1$ and $S_w = 8$.

Plot the graph of total time taken for file transfer against the Pc for $S_w = 1$ and $S_w = 8$

Constraint: P_{Err} is fixed to 0, MTU is fixed to 1000, T_{Out} is fixed 0.5s, file size is fixed to 1 MB.

Variables: S_w, P_c and response time for file transfer.

Goal: To find the behaviour of Go back N and stop and wait, which protocol is more sensitive to P_c.

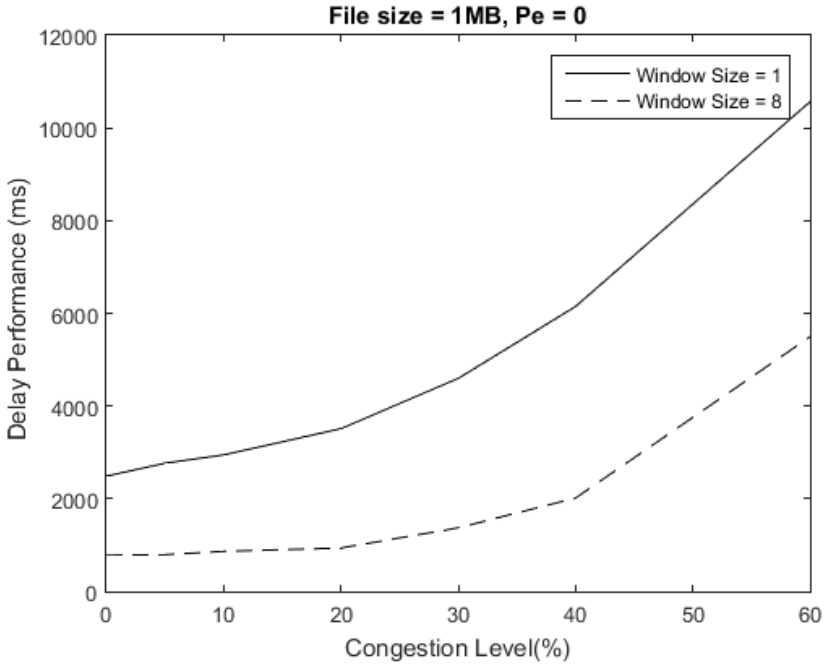

Figure 22: Go back N performance against Stop and Wait in congested channel

Conclusion: Based upon the graph, we can conclude that the sensitivity of Stop and Wait is comparatively equal , the gap between the plots increases slightly after 40% congestion level.

7.4 Go back N performs against Stop and wait in a Error prone and congested channel

Here we compare the performance of Go back N in a congested and error prone channel and the performance of Stop and Wait in similar condition and check the sensitivity of both the protocols.

Strategy: For a fixed size of file (1MB) we vary probability of error and percentage congestion and see how Go back N behaves and what is the cumulative effect on the protocol. Continue the experiment for $P_{Err} = 0$, $P_{Err} = 0.1$,$P_{Err} = 0.3$,$P_{Err} = 0.5$ and P_c= 0, P_c= 10, P_c= 20, P_c= 30, $P_c = 40$ and $P_c = 50$.

Repeat the experiment for stop and wait and collect the time readings.

Plot the graph of total time taken for file transfer against the P_c for a set of values of P_{Err}.

Constraint: S_w is fixed, MTU is fixed, T_{Out} is fixed, file size is fixed.

Variables: P_c , P_{Err} and total time taken for file transfer.

Goal: To find the behaviour of Go back N and the effect of both probability of error and percentage of congestion together and compare that to the Stop and wait performance.

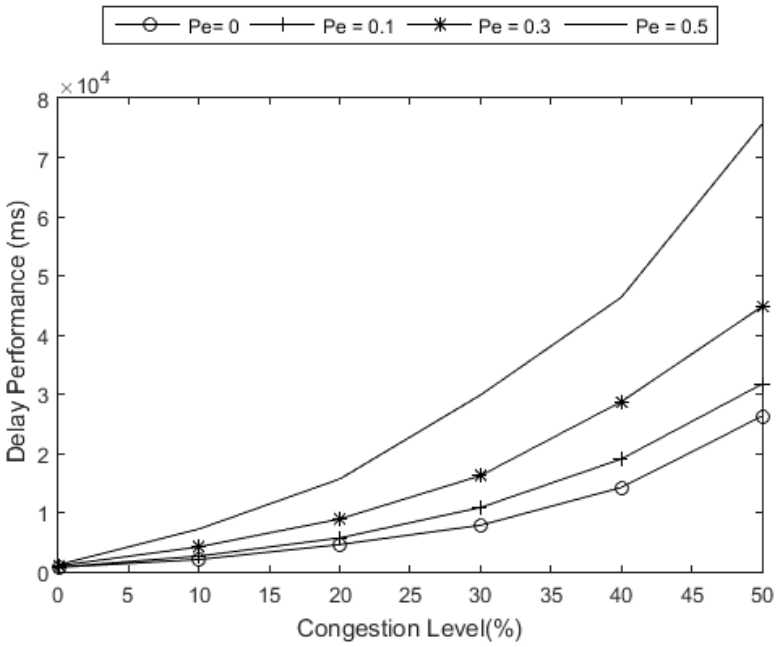

Figure 23: Go back N performance in Error prone and congested channel

Based on the collected readings of time, plot the graph for Stop and wait as well as Go back N of window size 8 and check the sensitivity.

Analysis: Comparing the plots of stop and wait and go back n, we can notice that the plots at $P_{Err} = 0.3$ the sensitivity remains almost same for both the protocols but at

Congestion Level(%)	0	10	20	30	40	50
Mean(ms)	1190.1	7295.2	15841.8	15714	48639	76977.2
STD	102.2	337.5	1066.6	1775.8	2764.8	3549.4
Confidence Interval(95%)	70.8	233.8	630.3	1230.5	1915.9	2459.6

Table 6: GBN: Error probability = 0.5, File Size = 1MB

Congestion Level(%)	0	10	20	30	40	50
Mean(ms)	2498.1	14528.5	27291.7	45139.6	68317.3	115314.6
STD	128.8	1451.6	2472.6	3206	4039.1	4645.9
Confidence Interval(95%)	89.3	1005.9	1713.4	2221.6	2798.9	3219.4

Table 7: Stop and Wait: Error probability = 0.5, File Size = 1MB

$P_{Err} = 0.5$ the plot for stop wait increases at a much higher rate when compared to go back n.
Calculating the standard deviation and confidence interval for both the protocols (refer the tables for actual values) also highlights the fact that standard deviation and the confidence window increases much faster in stop and wait when compared to go back n.

Conclusion: Based upon the graph, and the calculations the sensitivity of stop and wait is higher with the increasing value of error probability when compared with the Go back N. Gap between the curves increases with increase in error probability.

8 Project Time-line

Actual implementation time line:

Week 1: Implemented the basic structure of name server, file server and client, IP look from the system, reading name server info from the well-known file $name\_server\_info$ and writing into the name server file. Created the configuration file, listed the configurable items for the project and reading the configurable items from the configuration file.

Week 2: Implemented the CLI command structure, getting random NS info from the $name\_server\_info$ file, decided on the UDP structure and implemented, defined register, resolve, service update, $SFTP$ command methods, dynamic array [3] in name server for maintaining file server IP and port.

Week 3: Implemented method to register and synch between the name servers when there are more than 1 name server, establishing control connection with the FS, sending control packets between File server and client.

Week 4: Implemented the load balancing in the name server using the dynamic array [3], implemented the packing unpacking function [4] for packet structure Packing, made few changes in the flow of register , resolve, register NS and sync using packing and unpacking[4]. Fork implemented in file server in order to handle multiple clients at a

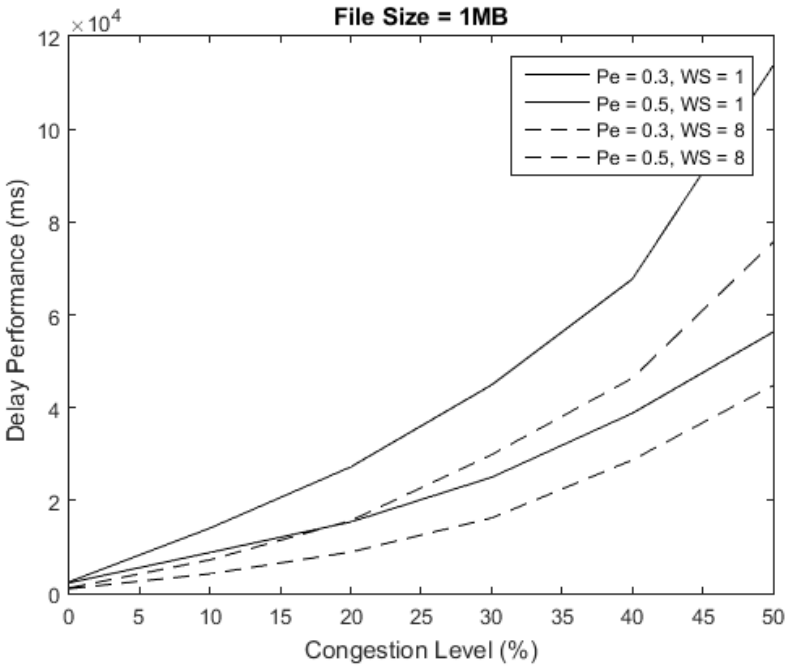

Figure 24: Go back N performance against Stop and Wait in Error prone and congested channel

time. Time out implemented for control connection between name server, file server and client.

Week 5: Implemented the data connection between the client and file server. Implemented data commands get, mget, put and mput commands. File transfer between client and file server. Designed and implemented Go back N protocol and CRC. Error injection and packet dropping implemented.

Week 6: Testing and bug fixing, statistical analysis and documentation.

9 Conclusion

Our main aim was to implement a simple file transfer protocol and analyse the performance of the implemented protocol. We were successful in testing the performance of Go back N and Stop and Wait protocol in our project as well. In future we would like to implement the RTP (Real Time Protocol) to distinguish between the media files and the documents and provide the Quality of Services (QoS) based on the file type. We would like to collect more information and try to implement over our design.

10 Bibliography

References

[1] *CCITT CRC 16 polynomial implementation.* URL: http://stjarnhimlen.se/snippets/crc-16.c.

[2] J.F.Kurose. *Alternating bit and GO-BACK-N network emulator.* URL: http://www.cise.ufl.edu/~mgargano/part2/prog2.c.

[3] *Learn C the hardway, Dynamic Array.* URL: http://c.learncodethehardway.org/book/ex34.html.

[4] *Macros for packing floats and doubles.* URL: http://beej.us/guide/bgnet/examples/pack2.c.

11 Appendix

```
oxygen{25} cd private/SFTP/SFTPNS
oxygen{26} make -f makefile SFTPNS
make: `SFTPNS' is up to date.
oxygen{27} SFTPNS
Control channel created...
Bootstrapping name server...
Created control channel for 136.142.227.10:7655
Updated Name Server Info with 0-136.142.227.10-7655
Ready to listen...
```

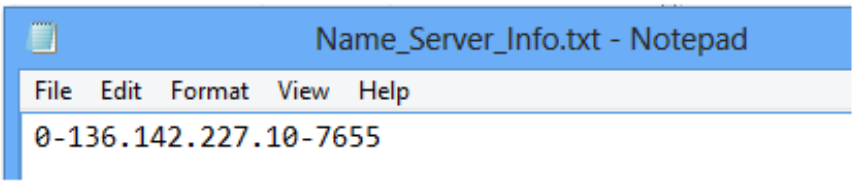

Figure 25: Name Server Booting

```
oxygen{25} cd private/SFTP/SFTPNS
oxygen{26} SFTPNS
Control channel created...
Bootstrapping name server...
Updated Name Server Info with 1-136.142.227.10-13197
Obtained neighbouring name server info - 136.142.227.10:7655...
Requesting Name Server to register... Attempt: 1
Registration to neighbouring name server is successful...
Ready to listen...
```

Figure 26: Secondary Name Server Booting

```
Ready to listen...
Received packet to service...
Serviced register name server request from 136.142.227.10:13197
Ready to listen...
```

Figure 27: Name Server NSRegister

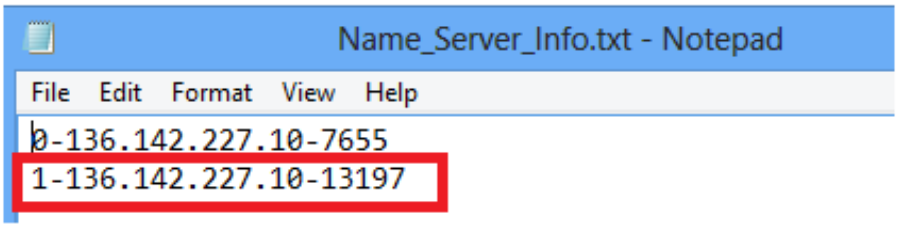

Figure 28: Name server info file

```
oxygen{27} SFTPFS
Control channel created...
Bootstrapping file server...
Created control channel for 136.142.227.10:46549
Obtained name server info - 136.142.227.10:7655...
Requesting Name Server to register... Attempt: 1
Registration to name server is successful...
Ready to listen...
```

Figure 29: File Server register

```
Ready to listen...
Received packet to service...
Serviced register file server request from 136.142.227.10:46549
Obtained neighbouring name server info - 136.142.227.10:13197...
Requesting Name Server to register... Attempt: 1
Synchronisation with neighbouring name server is successful...
Ready to listen...
```

Figure 30: Name Server view: File Server register

```
Ready to listen...
Received packet to service...
Serviced sync request from 136.142.227.10:7655
Ready to listen...
```

Figure 31: Name Server sync

```
oxygen{29} SFTPClient
_____

Welcome to SFTP!!!
_____

SFTP>
```

Figure 32: Run Client Application

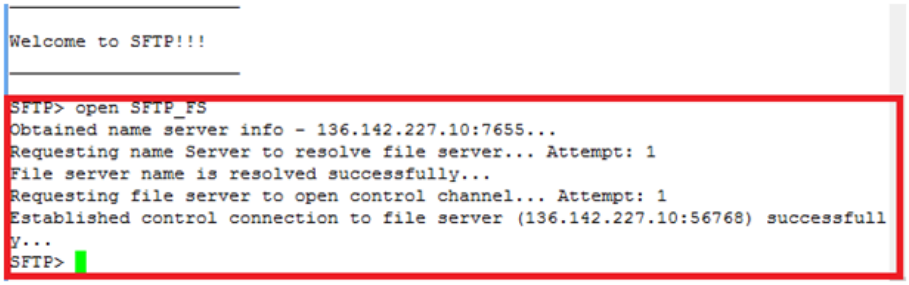

Figure 33: Open FSname

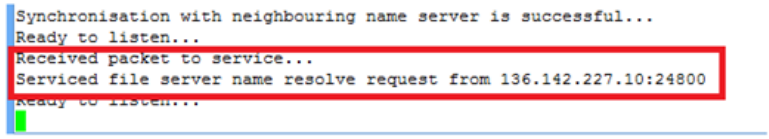

Figure 34: Resolve FS request

Figure 35: Client connect to File server

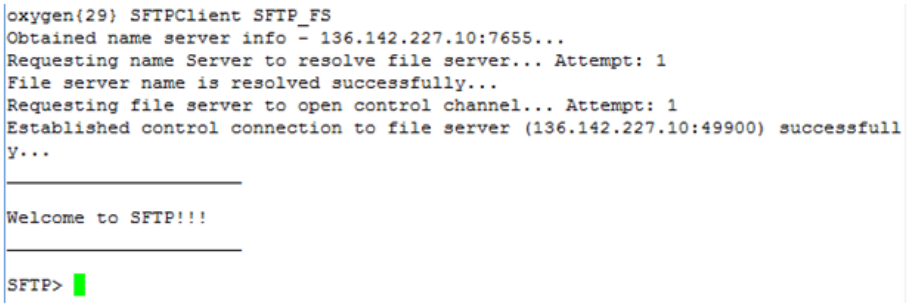

Figure 36: SFTP FSname

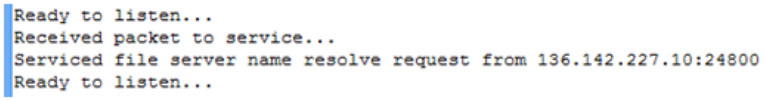

Figure 37: NS view: resolve request

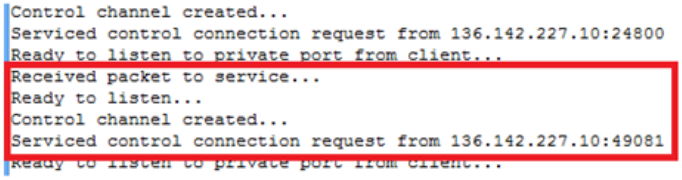

Figure 38: FS accepting control packets from Client

```
SFTP> lpwd

Current directory: /afs/cs.pitt.edu/usr0/darshan/private/SFTP/SFTPClient
SFTP> rpwd
Current directory at file server: /afs/cs.pitt.edu/usr0/darshan/private/SFTP/SFT
PFS
```

Figure 39: lpwd and rpwd

```
SFTP> lls

Total count: 368

drwxr-xr-x 1 darshan users       2048 Apr 07 10:32 Client
drwxr-xr-x 1 darshan users       2048 Apr 07 10:58 Global
-rw-r--r-- 1 darshan users       1132 Apr 07 08:00 makefile
-rw-r--r-- 1 darshan users        231 Apr 07 07:32 objects.mk
-rw-r--r-- 1 darshan users        463 Apr 07 08:00 sources.mk
-rwxr-xr-x 1 darshan users     356606 Apr 07 10:36 SFTPClient

SFTP>
```

Figure 40: lls command

```
SFTP> rls
Directory listing of current directory at file server...
drwxr-xr-x 1 darshan users       2048 Apr 07 10:49 FS
drwxr-xr-x 1 darshan users       2048 Apr 07 10:00 Global
-rw-r--r-- 1 darshan users        500 Mar 26 02:54 IPLookUp.c
-rw-r--r-- 1 darshan users       1112 Apr 07 08:02 makefile
-rw-r--r-- 1 darshan users        231 Apr 07 07:34 objects.mk
-rw-r--r-- 1 darshan users        459 Apr 07 08:02 sources.mk
-rwxr-xr-x 1 darshan users     349277 Apr 07 10:52 SFTPFS

SFTP>
```

Figure 41: rls command

```
SFTP> rcd /afs/cs.pitt.edu/usr0/darshan/private/Remote
rcd: Successful...
SFTP> rpwd
Current directory at file server: /afs/cs.pitt.edu/usr0/darshan/private/Remote
SFTP> rcd /invalid/directory
rcd: Failed...
SFTP>
```

Figure 42: rcd command

```
rcd: Failed...
SFTP> lcd /afs/cs.pitt.edu/usr0/darshan/private/Local

Local directory changed successfully...
SFTP> lpwd

Current directory: /afs/cs.pitt.edu/usr0/darshan/private/Local
SFTP> lcd /invalid/directory

Changing local directory failed...
SFTP> lpwd

Current directory: /afs/cs.pitt.edu/usr0/darshan/private/Local
SFTP>
```

Figure 43: lcd command

```
SFTP> lls

Total count: 128

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:12 FS_Errorlog.txt

SFTP> rls
Directory listing of current directory at file server...
-rw-r--r-- 1 darshan users      85812 Apr 03 21:01 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 02 20:48 WIN_20140611_153308.JPG

SFTP> get FS_Errorlog.txt
Control channel created...
Created data connection with 136.142.227.10:20124
Received packet to service...
File FS_Errorlog.txt already exists in local machine. Do you want to overwrite? Say (y)es or (n)o: n
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49900
SFTP>
```

Figure 44: get: duplicate file cancelled

```
SFTP> get FS_Errorlog.txt
Control channel created...
Created data connection with 136.142.227.10:53940
Received packet to service...
File FS_Errorlog.txt already exists in local machine. Do you want to overwrite? Say (y)es or (n)o: y
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file FS_Errorlog.txt from the remote machine...
File FS_Errorlog.txt received successfully...
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49900
SFTP>
```

Figure 45: get: duplicate file

```
SFTP> rls
Directory listing of current directory at file server...
-rw-r--r-- 1 darshan users      85812 Apr 03 21:01 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 02 20:48 WIN_20140611_153308.JPG

SFTP> lls

Total count: 128

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:25 FS_Errorlog.txt

SFTP> get invalid.file
Control channel created...
No matching file present in remote machine to transfer...
SFTP>
```

Figure 46: get invalid file

```
Ready to listen to private port from client...
Received packet to service
No matching file present in remote machine to transfer...
Ready to listen to private port from client...
```

Figure 47: Server view: get invalid file

```
SFTP> get WIN_20140611_153308.JPG
Control channel created...
Created data connection with 136.142.227.10:25494
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file WIN_20140611_153308.JPG from the remote machine...
File WIN_20140611_153308.JPG received successfully...
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49900
SFTP> rls
Directory listing of current directory at file server...
-rw-r--r-- 1 darshan users      85812 Apr 03 21:01 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 02 20:48 WIN_20140611_153308.JPG

SFTP> lls

Total count: 204

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:25 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 07 13:14 WIN_20140611_153308.JPG
```

Figure 48: Get command

```
SFTP> put WIN_20140611_153308.JPG
Control channel created...
Created data connection with 136.142.227.10:52157
File WIN_20140611_153308.JPG already exists in remote machine. Do you want to overwrite? Say (y)es or (n)o: n
SFTP>
SFTP> put WIN_20140611_153308.JPG
Control channel created...
Created data connection with 136.142.227.10:11439
File WIN_20140611_153308.JPG already exists in remote machine. Do you want to overwrite? Say (y)es or (n)o: y
Transferring file WIN_20140611_153308.JPG to the remote machine...
File WIN_20140611_153308.JPG sent successfully...
SFTP>
```

Figure 49: Client view: put over write

```
Received packet to service...
Serviced send file request from 136.142.227.10:49081
Received packet to service...
Serviced start transfer request from 136.142.227.10:49081
Receiving file WIN_20140611_153308.JPG from the remote machine...
File WIN_20140611_153308.JPG received successfully...
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49081
Ready to listen to private port from client...
```

Figure 50: Server view: put Command

```
SFTP> put WIN_20140611_153308.JPG
Control channel created...
Created data connection with 136.142.227.10:49353
File WIN_20140611_153308.JPG already exists in remote machine and cannot be overwritten...
SFTP> rls
Directory listing of current directory at file server...
-r--r--r-- 1 darshan users     127906 Apr 07 13:59 WIN_20140611_153308.JPG
-rw-r--r-- 1 darshan users      85812 Apr 03 21:01 FS_Errorlog.txt
```

Figure 51: File over write failed: No Write Access

```
SFTP> rls
Directory listing of current directory at file server...
-r--r--r-- 1 darshan users     127906 Apr 07 13:59 WIN_20140611_153308.JPG
-rw-r--r-- 1 darshan users      85812 Apr 03 21:01 FS_Errorlog.txt

SFTP> lls

Total count: 204

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:25 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 07 13:14 WIN_20140611_153308.JPG

SFTP> mget *.*
Control channel created...
Created data connection with 136.142.227.10:4566
Received packet to service...
File WIN_20140611_153308.JPG already exists in local machine. Do you want to overwrite? Say (y)es or (n)o: y
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file WIN_20140611_153308.JPG from the remote machine...
File WIN_20140611_153308.JPG received successfully...
Received packet to service...
File FS_Errorlog.txt already exists in local machine. Do you want to overwrite? Say (y)es or (n)o: n
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49900
SFTP>
SFTP> lls

Total count: 204

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:25 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 07 13:56 WIN_20140611_153308.JPG
```

Figure 52: mget *.*

```
SFTP> mget T*.*
Control channel created...
Created data connection with 136.142.227.10:45978
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file TCP_Frame.png from the remote machine...
File TCP_Frame.png received successfully...
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file TCP_Frame_Ack.png from the remote machine...
File TCP_Frame_Ack.png received successfully...
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49900
```

Figure 53: mget T*.*

```
SFTP> mget *.txt
Control channel created...
Created data connection with 136.142.227.10:15332
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file test.txt from the remote machine...
File test.txt received successfully...
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file sam.txt from the remote machine...
File sam.txt received successfully...
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file FAQs in Unix.txt from the remote machine...
File FAQs in Unix.txt received successfully...
Received packet to service...
File Ger.txt already exists in local machine. Do you want to overwrite? Say (y)es or (n)o: n
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49900
```

Figure 54: mget *.txt

```
-rw-r--r-- 1 darshan users       9811 Mar 30 21:59 Packet structure.docx
-rw-r--r-- 1 darshan users       6000 Apr 07 02:16 sam.txt
-rw-r--r-- 1 darshan users      79052 Mar 30 21:04 FAQs in Unix.txt
-rw-r--r-- 1 darshan users       9070 Jan 11 06:04 Probability.xlsx
-rw-r--r-- 1 darshan users       4761 Feb 10 17:36 TCP_Frame.png
-rw-r--r-- 1 darshan users       4150 Feb 10 17:06 TCP_Frame_Ack.png
-rw-r--r-- 1 darshan users     127906 Jun 11 06:08 WIN_20140611_153308.JPG
-rw-r--r-- 1 darshan users      40000 Feb 16 16:44 10.1.1.27.869.pdf
-rw-r--r-- 1 darshan users     277252 Mar 31 13:02 test.pdf
-rw-r--r-- 1 darshan users    3025673 Mar 31 23:46 Ger.txt
-rw-r--r-- 1 darshan users     194793 Feb 16 16:22 FTP_WP_acc_IM_AE.PDF
-rw-r--r-- 1 darshan users     122643 Feb 16 16:28 MIT16_36s09_lec18.pdf
-rw-r--r-- 1 darshan users      78029 Apr 02 14:40 DH03.pdf
-rw-r--r-- 1 darshan users     290947 Feb 14 15:36 E1064102512.pdf
-rw-r--r--
SFTP> lls

Total count: 204

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:25 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 07 13:56 WIN_20140611_153308.JPG

SFTP> mget Probability.xlsx,Packet structure.docx,10.1.1.27.869.pdf
Control channel created...
Created data connection with 136.142.227.10:10895
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file Packet structure.docx from the remote machine...
File Packet structure.docx received successfully...
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file Probability.xlsx from the remote machine...
File Probability.xlsx received successfully...
Received packet to service...
Serviced send file request from 136.142.227.10:49900
Received packet to service...
Serviced start transfer request from 136.142.227.10:49900
Receiving file 10.1.1.27.869.pdf from the remote machine...
File 10.1.1.27.869.pdf received successfully...
Received packet to service...
Serviced data connection termination request from 136.142.227.10:49900
```

Figure 55: mget file names

```
SFTP> rls
Directory listing of current directory at file server...
-rw-r--r-- 1 darshan users    47145598 Aug 17 02:33 sample.mp4
-rw-r--r-- 1 darshan users       38200 Apr 06 13:00 sam.txt
-rw-r--r-- 1 darshan users   202269258 Dec 16 06:29 VIDEO0015.mp4

SFTP> lls

Total count: 743

-rw-r--r-- 1 darshan users     3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users       85812 Apr 07 13:25 FS_Errorlog.txt
-rw-r--r-- 1 darshan users      127906 Apr 07 13:56 WIN_20140611_153308.JPG
-rw-r--r-- 1 darshan users        9811 Apr 07 14:54 Packet structure.docx
-rw-r--r-- 1 darshan users        9070 Apr 07 14:54 Probability.xlsx
-rw-r--r-- 1 darshan users       40000 Apr 07 14:54 10.1.1.27.869.pdf
-rw-r--r-- 1 darshan users    33142403 Apr 07 14:57 test.txt
-rw-r--r-- 1 darshan users        6000 Apr 07 14:57 sam.txt
-rw-r--r-- 1 darshan users       79052 Apr 07 14:57 FAQs in Unix.txt
-rw-r--r-- 1 darshan users        4761 Apr 07 14:53 TCP_Frame.png
-rw-r--r-- 1 darshan users        4150 Apr 07 14:53 TCP_Frame_Ack.png

SFTP> mput FAQs in Unix.txt,Probability.xlsx
Control channel created...
Created data connection with 136.142.227.10:18922
Transferring file Probability.xlsx to the remote machine...
File Probability.xlsx sent successfully...
Transferring file FAQs in Unix.txt to the remote machine...
File FAQs in Unix.txt sent successfully...
```

Figure 56: mput file names

```
SFTP> mput T*.*
Control channel created...
Created data connection with 136.142.227.10:38027
Transferring file TCP_Frame.png to the remote machine...
File TCP_Frame.png sent successfully...
Transferring file TCP_Frame_Ack.png to the remote machine...
File TCP_Frame_Ack.png sent successfully...
SFTP> rls
Directory listing of current directory at file server...
-rw-r--r-- 1 darshan users   47145598 Aug 17 02:33 sample.mp4
-rw-r--r-- 1 darshan users      38200 Apr 06 13:00 sam.txt
-rw-r--r-- 1 darshan users       4761 Apr 07 14:29 TCP_Frame.png
-rw-r--r-- 1 darshan users  202269258 Dec 16 06:29 VIDEO0015.mp4
-rw-r--r-- 1 darshan users       9070 Apr 07 14:23 Probability.xlsx
-rw-r--r-- 1 darshan users      79052 Apr 07 14:23 FAQs in Unix.txt
-rw-r--r-- 1 darshan users       4150 Apr 07 14:29 TCP_Frame_Ack.png

SFTP> lls

Total count: 743

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:25 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 07 13:56 WIN_20140611_153308.JPG
-rw-r--r-- 1 darshan users       9811 Apr 07 14:54 Packet structure.docx
-rw-r--r-- 1 darshan users       9070 Apr 07 14:54 Probability.xlsx
-rw-r--r-- 1 darshan users      40000 Apr 07 14:54 10.1.1.27.869.pdf
-rw-r--r-- 1 darshan users   33142403 Apr 07 14:57 test.txt
-rw-r--r-- 1 darshan users       6000 Apr 07 14:57 sam.txt
-rw-r--r-- 1 darshan users      79052 Apr 07 14:57 FAQs in Unix.txt
-rw-r--r-- 1 darshan users       4761 Apr 07 14:53 TCP_Frame.png
-rw-r--r-- 1 darshan users       4150 Apr 07 14:53 TCP_Frame_Ack.png
```

Figure 57: mput T*.*

```
SFTP> mput *.*
Control channel created...
Created data connection with 136.142.227.10:40382
Transferring file Ger.txt to the remote machine...
File Ger.txt sent successfully...
Transferring file FS_Errorlog.txt to the remote machine...
File FS_Errorlog.txt sent successfully...
Transferring file WIN_20140611_153308.JPG to the remote machine...
File WIN_20140611_153308.JPG sent successfully...
Transferring file Packet structure.docx to the remote machine...
File Packet structure.docx sent successfully...
File Probability.xlsx already exists in remote machine. Do you want to overwrite? Say (y)es or (n)o: y
Transferring file Probability.xlsx to the remote machine...
File Probability.xlsx sent successfully...
Transferring file 10.1.1.27.869.pdf to the remote machine...
File 10.1.1.27.869.pdf sent successfully...
Transferring file test.txt to the remote machine...
File test.txt sent successfully...
File sam.txt already exists in remote machine. Do you want to overwrite? Say (y)es or (n)o: n
File FAQs in Unix.txt already exists in remote machine. Do you want to overwrite? Say (y)es or (n)o: n
File TCP_Frame.png already exists in remote machine. Do you want to overwrite? Say (y)es or (n)o: n
File TCP_Frame_Ack.png already exists in remote machine. Do you want to overwrite? Say (y)es or (n)o: y
Transferring file TCP_Frame_Ack.png to the remote machine...
File TCP_Frame_Ack.png sent successfully...
SFTP>
SFTP> rls
Directory listing of current directory at file server...
-rw-r--r-- 1 darshan users   47145598 Aug 17 02:33 sample.mp4
-rw-r--r-- 1 darshan users      38200 Apr 06 13:00 sam.txt
-rw-r--r-- 1 darshan users       4761 Apr 07 14:29 TCP_Frame.png
-rw-r--r-- 1 darshan users  202269258 Dec 16 06:29 VIDEO0015.mp4
-rw-r--r-- 1 darshan users       9070 Apr 07 14:45 Probability.xlsx
-rw-r--r-- 1 darshan users      79052 Apr 07 14:23 FAQs in Unix.txt
-rw-r--r-- 1 darshan users       4150 Apr 07 14:19 TCP_Frame_Ack.png
-rw-r--r-- 1 darshan users    3025673 Apr 07 14:42 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 14:42 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 07 14:42 WIN_20140611_153308.JPG
-rw-r--r-- 1 darshan users       9811 Apr 07 14:42 Packet structure.docx
-rw-r--r-- 1 darshan users      40000 Apr 07 14:45 10.1.1.27.869.pdf
-rw-r--r-- 1 darshan users   33142403 Apr 07 14:11 test.txt

SFTP> lls
```

Figure 58: mput *.*

```
Total count: 743

-rw-r--r-- 1 darshan users    3025673 Apr 07 10:23 Ger.txt
-rw-r--r-- 1 darshan users      85812 Apr 07 13:25 FS_Errorlog.txt
-rw-r--r-- 1 darshan users     127906 Apr 07 13:56 WIN_20140611_153308.JPG
-rw-r--r-- 1 darshan users       9811 Apr 07 14:54 Packet structure.docx
-rw-r--r-- 1 darshan users       9070 Apr 07 14:54 Probability.xlsx
-rw-r--r-- 1 darshan users      40000 Apr 07 14:54 10.1.1.27.869.pdf
-rw-r--r-- 1 darshan users   33142403 Apr 07 14:57 test.txt
-rw-r--r-- 1 darshan users       6000 Apr 07 14:57 sam.txt
-rw-r--r-- 1 darshan users      79052 Apr 07 14:57 FAQs in Unix.txt
-rw-r--r-- 1 darshan users       4761 Apr 07 14:53 TCP_Frame.png
-rw-r--r-- 1 darshan users       4150 Apr 07 14:53 TCP_Frame_Ack.png
```

Figure 59: mput *.*: overwrite cancelled

```
SFTP> close
Disconnecting connection to file server...
Connection to the server is terminated...
SFTP>
```

Figure 60: Client view: Close command

```
Serviced SFTP control command request from 136.142.227.10:49081
Ready to listen to private port from client...
Received packet to service...
Serviced closing control connection request from 136.142.227.10:49081
```

Figure 61: Server view: Close command

```
SFTP> open SFTP_FS
Obtained name server info - 136.142.227.10:7655...
Requesting name Server to resolve file server... Attempt: 1
File server name is resolved successfully...
Requesting file server to open control channel... Attempt: 1
Established control connection to file server (136.142.227.10:61086) successfully...
SFTP>
```

Figure 62: OPEN Command

```
Serviced closing control connection request from 136.142.227.10:49081
Received packet to service...
Ready to listen...
Control channel created...
Serviced control connection request from 136.142.227.10:49081
Ready to listen to private port from client...
```

Figure 63: Close Open scenario

```
SFTP> close
Disconnecting connection to file server...
Connection to the server is terminated...
SFTP> open SFTP_FS
Obtained name server info - 136.142.227.10:7655...
Requesting name Server to resolve file server... Attempt: 1
File server name is resolved successfully...
Requesting file server to open control channel... Attempt: 1
Established control connection to file server (136.142.227.10:61086) successfully...
SFTP> open SFTP_FS
You are already connected to the server...
SFTP> close
Disconnecting connection to file server...
Connection to the server is terminated
SFTP> rls
You are not yet connected to the server...
SFTP>
```

Figure 64: Open and close command

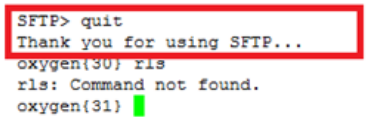

Figure 65: Quit when disconnected to FS

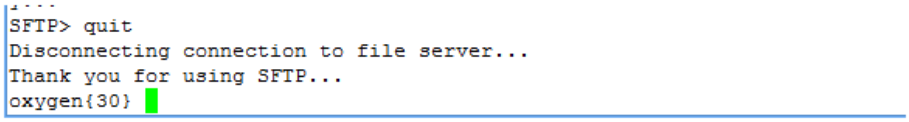

Figure 66: Quit when connected to FS

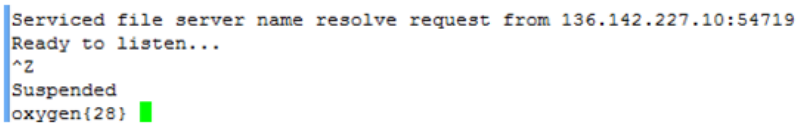

Figure 67: Name Server 1 Down

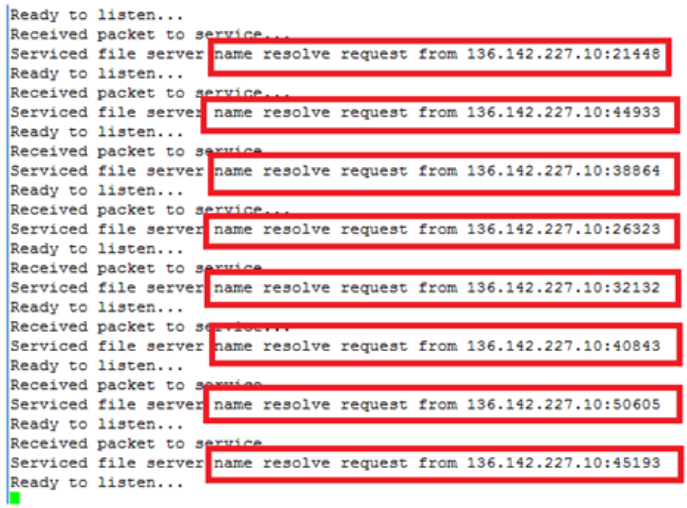

Figure 68: Name Server:Resolve request

Figure 69: FS accepting request from clients

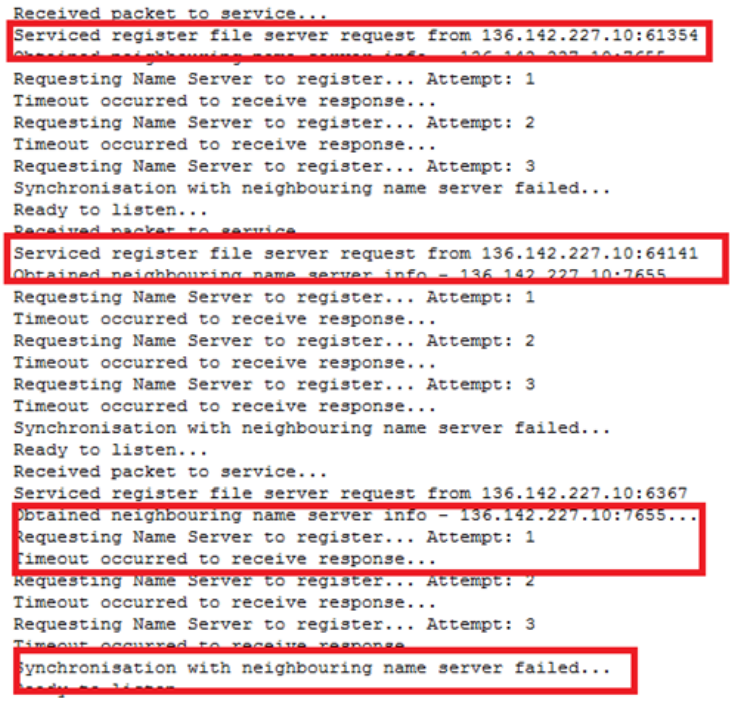

Figure 70: NS2 attempt to Sync

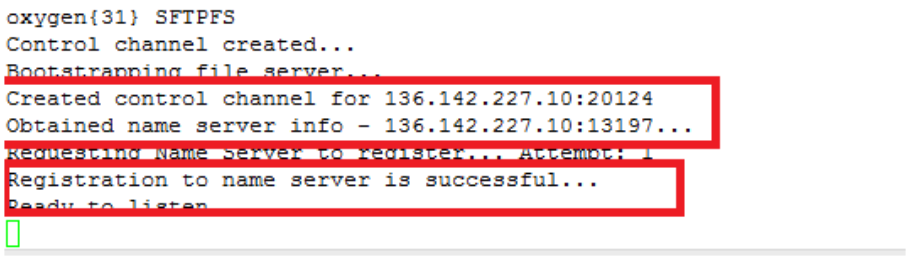

Figure 71: Register success at first attempt

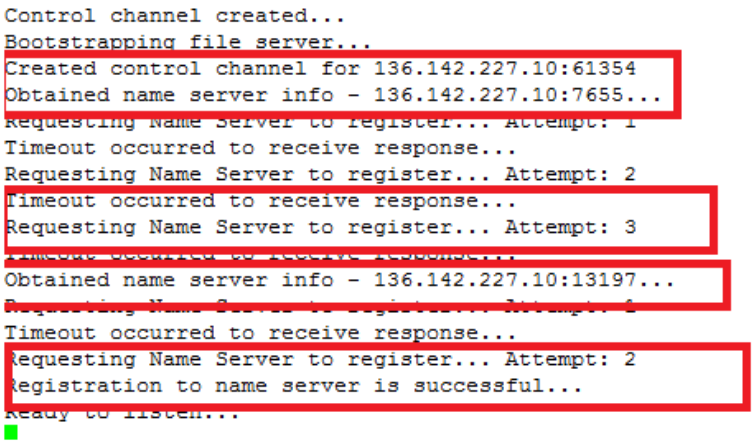

Figure 72: Attempts to register request

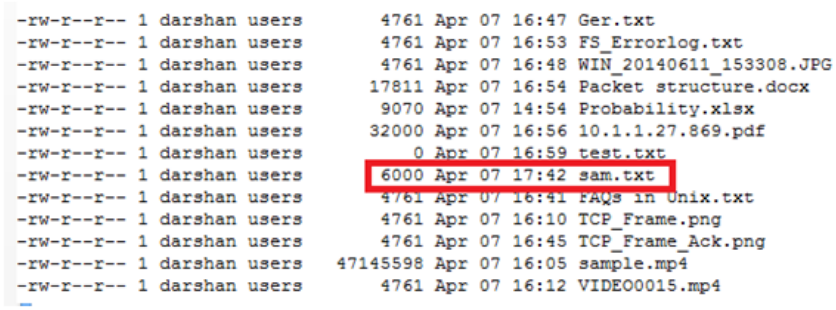

Figure 73: Client1 reading same files

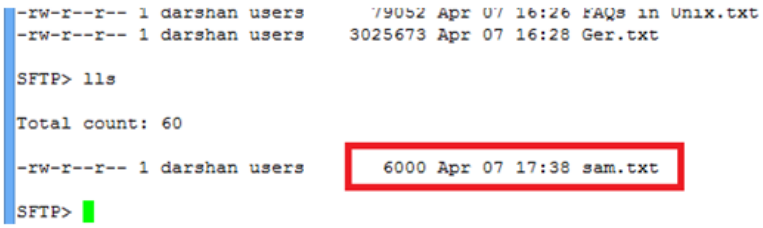

Figure 74: Client2 reading same files

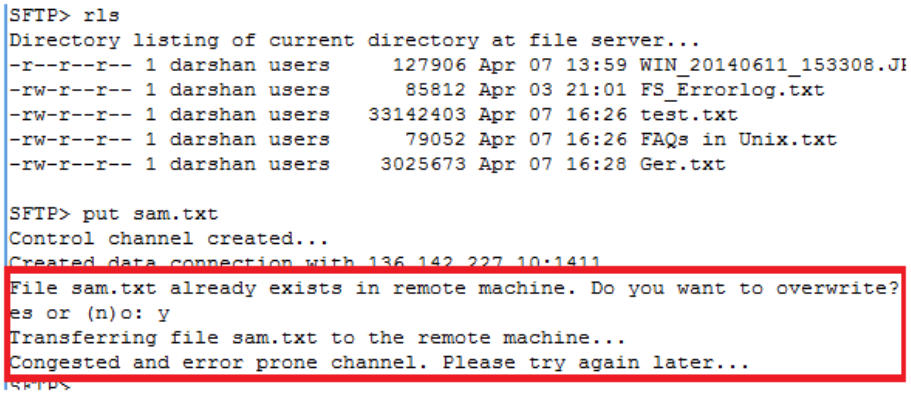

Figure 75: Client1 fails to write

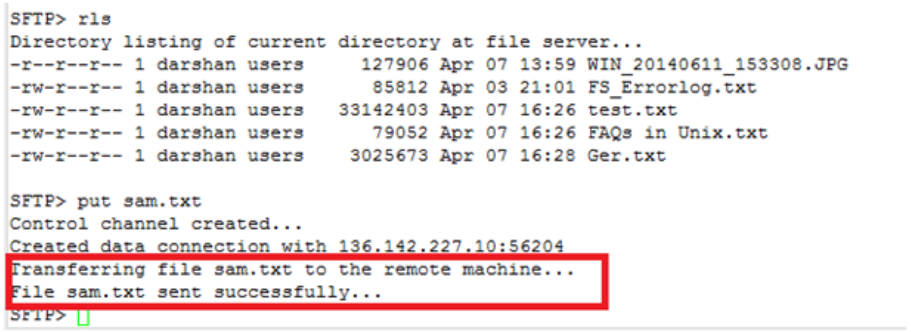

Figure 76: Client2 succeeds to write

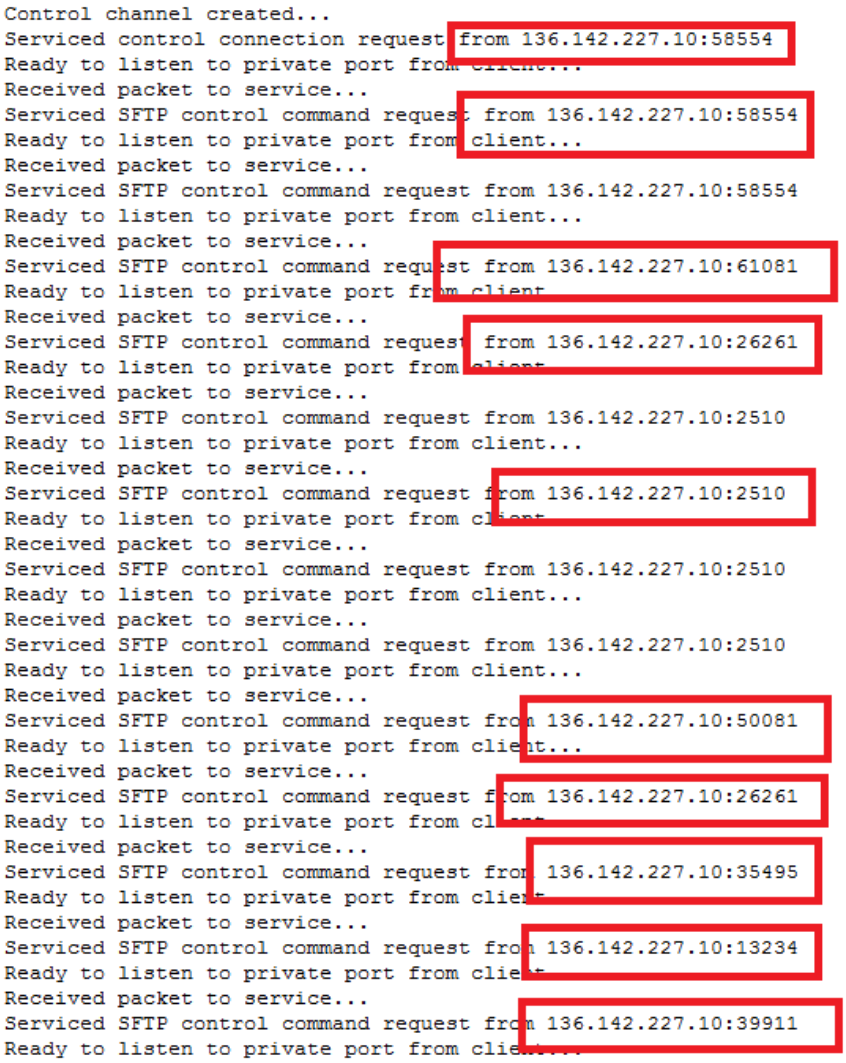

Figure 77: Multiple Clients connect to File server

Figure 78: Multiple Clients File transfer handled by File server