UNIVERSITY OF PITTSBURGH

COMPUTER SCIENCE

CS 2520 - WANS

Simple File Transfer Protocol

Design Document

Authors:
Debashis Ganguly
Darshan Shetty

Supervisor:
Dr. Taieb Znati

February 10, 2015

Contents

1 Introduction

The document in review serves as a design document 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 to be developed 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. High level estimation and time-line for implementation are also captured in this document.

The goal of the project, that this document deals with, is to design, implement, 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.

In Section 2 of this document, a high-level identification and brief definition of major components can be found along with the layered architecture of the protocol where these components act upon and interplay. Whereas Section 3 is concerned with details of the items described in Section 2. In Section 3, the detailed architecture of components and sub-components, underlying design principles and choices, protocol and its layering, network connections between systems are captured in details. The configurable items are described in Section 4. Section 5 speaks of tools, programing platform and other requirements for the implementation. Detailed test strategies and statistical methods are described in Section 6. The final section, i.e., Section 7 tries to capture high-level milestones for the implementation with estimated effort.

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*
NC: Number of active concurrent connections between clients and a single File Server
n_T: Number of try
T_{Out}: Time out period

TCP: *Tranmission Control Protocol*
UDP: *User Datagram Protocol*
d_{Proc}**:** *Processing Delay*
d_{Prop}**:** *Propagation Delay*
d_{Trans}**:** *Transmission Delay*
Std: *Standard Deviation*
S_P**:** *Packet Size in bytes*
S_W**:** *Window Size*
p_{Err}**:** *Probability of packet in error*
P_C**:** *Percentage Congestion*
F_{NS}**:** *Fully qualified path for Name Server 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 are planning to provide the user with a command-line interface (CLI) or character user interface (CUI) at first and if time permits we will extend the same to provide a graphical user interface (GUI).*

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 hash 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.

Design 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.

Design Principle of FS and Design Choice:

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.

Once a FS is done servicing a Client and the control connection is terminated, it sends a control signal to NS and then NS decrements the counter for servicing clients. NS also periodically sends a health check monitoring signal to the NS to check whether it is active or in fault. Upon trying for a given number of time and waiting for a certain timeout period, NS declares a FS dead and removes it's entry from Hash Table and no longer redirect any resolve request of client to the file server.

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 to and at the same time it also sends a monitoring signal to NS to remove the former FS from the list being unresponsive and invariably dead.

3.2 Component Life Cycle and Interplay

3.2.1 Bootstrapping Name Sever

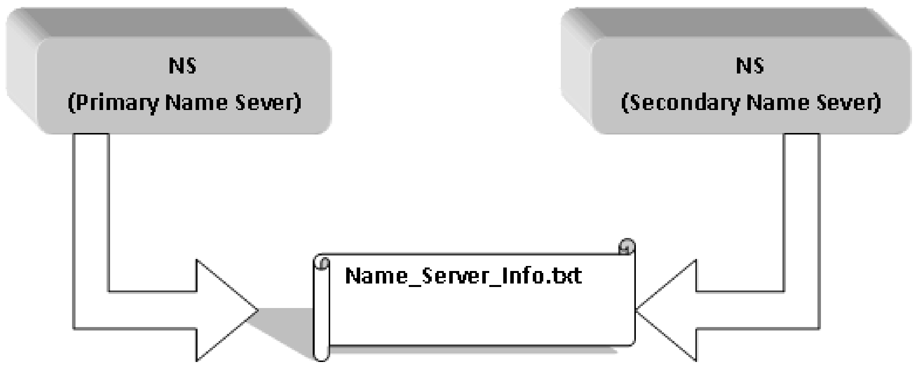


Figure 1: Bootstrapping Name Server

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

3.2.2 Bootstrapping File Servers and Synchronization between Name Servers

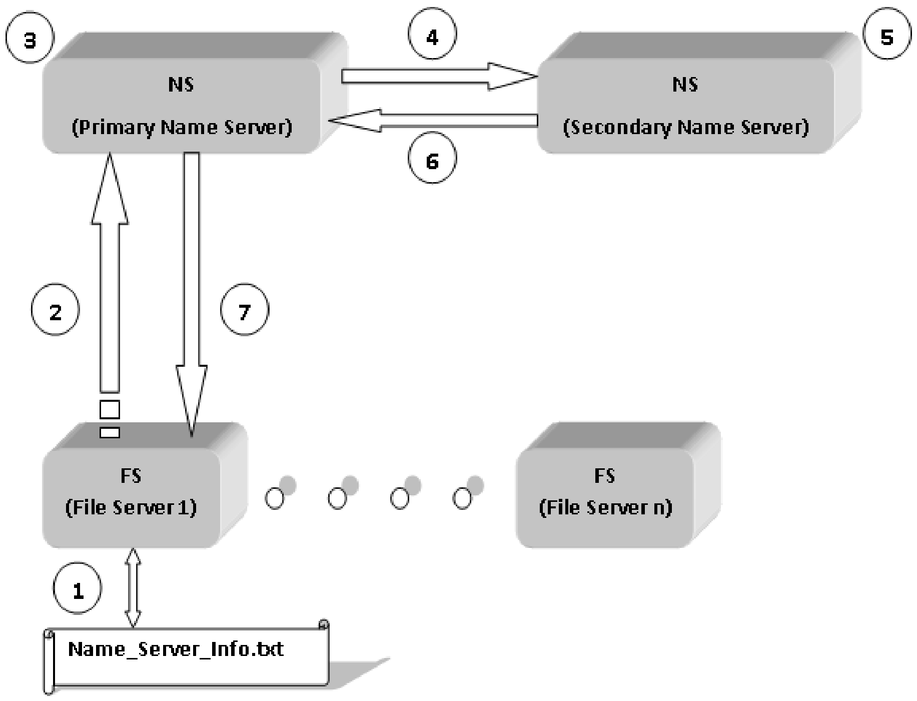


Figure 2: Registration of File Server to Name Server

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.

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

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*.

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

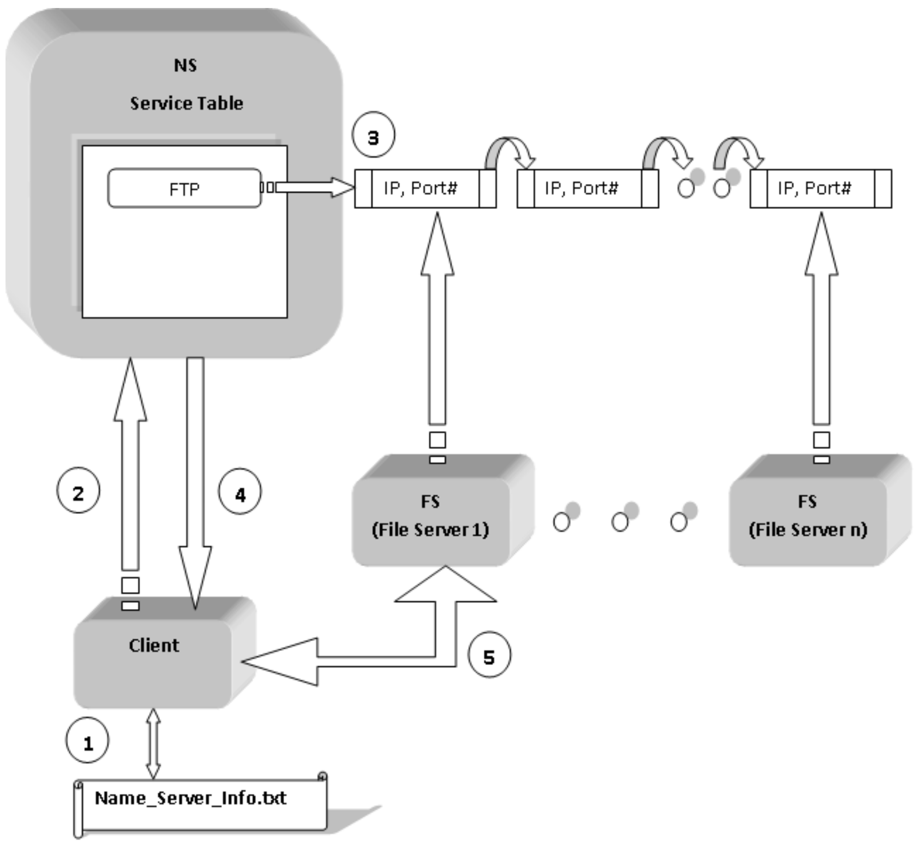


Figure 3: Resolving Query from Client by Name Server

To bootstrap the Client(s), the software component at Client is run to initialize the machine. As again we are not hard-coing 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 hash table against logical naming. Out of all listed FS, NS chooses a FS in round-robin fashion and increment the counter for servicing client by one. If a server is heavily loaded beyond a certain threshold, it passes along to choose the next available FS. Thus the NS also acts as a *Load Balancer* for FS.

Step 4: NS then sends IP and Port Number of the NS to client.

Step 5: Upon receiving the IP and Port Number, Client then establishes a connection with the FS.

3.2.4 Interaction between Client and Server

Connection and interaction between client and FS is on demand.

Step 1: Upon receiving the IP and Port Number from NS after resolve, Client establishes a connection with the FS.

Step 2: Client and Server establish a control connection to exchange commands and control signals over it

Step 3: When the client is ready to share a file, it requests a data connection with the server to share data packets.

Step 4: After the data transfer is over, data connection is terminated.

Step 5: The control connection is terminated only once the session between Client and Server is to be closed.

3.3 Protocol Layering in Details

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

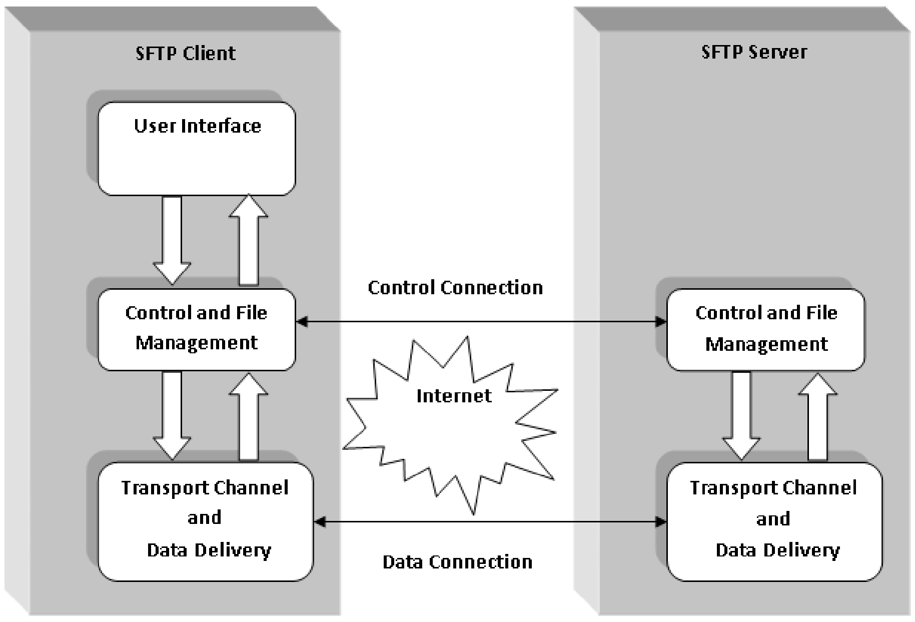

SFTP Client
SFTP Server
User Interface
Control and File Management
Control Connection
Control and File Management
Internet
Transport Channel and Data Delivery
Transport Channel and Data Delivery
Data Connection

Figure 4: 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.

2. *OPENFSName*: If the user doesn't provide host name as part of SFTP connection, then user specifies this command tp connect to the FS.
3. *CLOSE*: This command is used to terminate the connection between Client and the FS.
4. *ABORT*: This command cancels any immediate command being performed.
5. *QUIT*: This is issued to exit the SFTP environment.

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.
2. *LCD LocalDirectoryName*: To change directory at the local machine, doesn't require client to establish any connection with the server.
3. *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.
4. *PWD*: To show the present working directory at the remote machine (FS), control connection must be established between client and server before issuing this command.
5. *RCD RemoteDirectoryName*: To change directory at the remote machine (FS), control connection must be established between the client and server before issuing the command.

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.
2. *GET*: Request to copy the specified file from the FS to Client Local Machine.
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.
4. *PUT*: Request to copy the specified file from Client Local Machine to the FS.

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.

2. **Bad Directory Request:** Much similar to the above exception as part RCD if the asked directory is not valid, effective message is sent to UI.

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 *flock()* system call for the first client being serviced and the second user can be notified the same.

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. So to tackle this a soft warning can be raised to UI explaining the situation and providing the user whether to continue or postpone for later.

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:

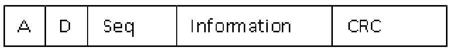
The first responsibility of this layer is to process the file sent by CFM to handle different aspects like as followings.

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.

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.

3. Variable file size and Reconstruction of file after decapsulation: How to handle variable file size, as both the packet and window size will be fixed between the sender and receiver. This layer has to rearrange the segments to reconstruct the file before passing it to upper layer. As the file size grows and considering scalability of system to support multiple clients, we cannot keep them in buffer in primary memory, rather they are written directly in temporary file location chunk by chunk before reconstruction of whole file.

4. Variable file size and Segmentation: It has to identify the file size and break it into number segments with the header information like sequence number, CRC bits in each segment and passes the segments to TC data sockets. The first of all segment will contain information like file size, directory name, file name etc and the last one will have an end of file information to denote completion. Number of segments is determined by file size, packet size, overhead size etc.



A	D	Seq	Information	CRC

Figure 5: Structure of Segment

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.

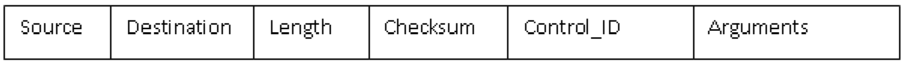
1. Control Connection:

   This connection is long lived and established between any OPEN command and CLOSE command. This channel is used to transmit control information like commands, ACKs, NAKs, Monitoring signal, Exceptions and Warning information and other signals to effectively manage the system.

   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.

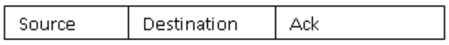
   Every commands or control packets are assigned an unique identifier to differentiate between each other, may be a byte to uniquely identify any item.

   The structure of packets used by this connection look like below.



Source	Destination	Length	Checksum	Control_ID	Arguments

Figure 6: Structure of Data for UDP



Source	Destination	Ack

Figure 7: Structure of Acknowledgement for UDP

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. Thus the best design choice to implement this is by using TCP.

   The structure of packets used by this connection look as below.

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 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

Source	Destination	Sequence	E	R	Window Size	Data	CRC

Figure 8: Structure of Frame for TCP

N	Sequence	Window Size

Figure 9: Structure of Acknowledgement for TCP Frame

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.

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 for each frame starts the timer.

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 NAK(n) (negative Ack) for the frame which was not received explicitly.

7. The sender has to resend from Frame number n onwards after receiving the NAK.

We are planning to implement 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.

Calculation of the time out:
$T_{Out} = 2 * d_{Prop} + 2 * d_{Proc} + d_{Trans_D} + d_{Tran_A} + Std$

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 are planning to place 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.

S_P**:** *Packet Size in bytes*
S_W**:** *Window Size*
p_{Err}**:** *Probability of packet in error*
P_C**:** *Percentage Congestion*
F_{NS}**:** *Fully qualified path for Name Server file*
NC: Number of active concurrent connections between clients and a single File Server
n_T: Number of try
T_{Out}: Time out period

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 to be developed is designed to be used in ***UNIX*** or ***LINUX*** platform and if time permits we will try to make it portable in Windows platform as well.

Language for Implementation:

We are going to implement 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 as this one 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 Analysis of Proposed System and Protocol

6.1 Test Strategy

As part of test strategy we are going to test components individually and independently and also perform 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 these 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.

6.1.1 Name Server Robustness Testing

1. Make primary name server down, and see if the secondary name server handles the load efficiently.
2. Make primary name server up, and check if the name server initialization runs and the servers are updated in the hash table.
3. Make a file server down, and check if the server name and IP is deleted from the NS hash table and no client request is forwarded to it.

6.1.2 Stress and load testing on the file server

Test the number of clients a server can handle, execute the test script which runs in a loop and executes a number of client application in different machines to connect to the same file server until the server raises an exception that it can not handle more clients. .

6.1.3 File Server and Client Concurrent access testing

Make multiple clients write into the same file on the server and test if file access exceptions are captured.

6.2 Statistical Analysis

6.2.1 Confidence Interval

Make a set of folders each containing different file sizes and configure different Error Probability P_{err} for eachh folder, run the test script for each folder. Every successful response, plot the Round Trip Time RTT and increase the window size by 1 till the maximum window size of the receiver has reached.

$RTT = 2 * d_{proc} + 2 * d_{prop} + d_{TransP} + d_{TransA}$

Considering processing delay d_{proc} and transmission delay for Ack d_{TransA} to bevery small, we can say $RTT = 2 * d_{prop} + d_{TransP}$

Plot the graph of RTT against window size S_W for each of the P_{Err} using MATLAB.

Calculate the confidence interval for window size 1 (stop and wait) and check for the optimal window size using the plot.

6.2.2 Performance of Stop and Wait against Go back N protocol

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.

We calculate the throughput efficiency η_{eff} against the window size S_W for different congestion percentage P_C.

Analyse how the stop and wait performs against sliding window protocol. Below are the calculations to perform to η_{eff}.

Consider,
R =Link Rate
n_f =Frame size
n_a = Ack size

Then the round trip time delay,
$RTT = 2 * d_{prop} + \frac{n_f}{R} + \frac{n_a}{R}$

Efficiency when error is 0,
$\eta_{eff=0} = \frac{1}{1+2*a}$, where $a = \frac{d_{prop}}{n_f/R}$

With error probability P_{err}

$\eta_p = \frac{(1-P_{err})}{(1+2a)((1-P_{err})+P_{err}*O/RTT)}$

where O is timeout.

7 Project Time-line

Project implementation is expected to be completed in 7 weeks including the test and analysis. The break up and the estimated time line are as below:

Week 1 (14 hours) estimated work to finish Client CLI UI implementation, Command definitions and primitive functions, Name Server Hash table and data structure.

Week 2 (10 hours) estimated work to finish CFM Command interpreter, mapping with the primitive functions, CFM file manager in client and file server.

Week 3 (5 hours) estimated work to finish CFM layer, segmentation and encapsulation.

Week 4 (10 hours) estimated work to finish CFM layer reconstruction, de-capsulation, primitive function definitions and implementation.

Week 5 (25 hours) estimated work to finish Data connection and Control connection, Transport layer socket programming, Boot strap programs for client, server and name server.

Week 6 (14 hours) estimated work to finish Test the interaction between all the components.

Week 7 (16 hours) estimated work to finish Testing and analysis, test scripts, Statistical data collection and analysis using MATLAB.

Bibliography

1. https://www.ietf.org/rfc/rfc959.txt
2. http://www.geeksforgeeks.org/little-and-big-endian-mystery/
3. http://www.networksorcery.com/enp/protocol/tcp.htm
4. http://www.freesoft.org/CIE/Course/Section4/8.htm
5. Computer Networking: A Top down approach – Kurose Ross