A Local/Remote File Manager Using File Transfer Protocol4Android

Sindhu A
Dept. of ISE
City Engineering College
Bangalore-560061
sindu.sbm@gmail.com

Bhavya S
Dept. of ISE
City Engineering College
Bangalore-560061
bhavyakshivanna@gmail.com

Abstract

Now a days it is common practice to handle any type of file with the personal computer. The introduction of mobile devices in modern life opened the doors to the possibility to do that ubiquitously, fostering a vast plethora of new entertainment applications. Unfortunately, the storage capacity of these devices is limited; it may hence be useful to have the possibility to store the files on the web and retrieve them at any time and anywhere. To satisfy this need, we have created FTP4Android. Our solution provides smartphone users with the illusion of having an infinite memory on their devices by storing their files on remote servers. To speed up the transfer process both in upload and download, parallel transmissions to/from different servers are performed. To demonstrate how this technology could be used for entertainment purposes, we have devised a treasure hunt application for mobile users based on our FTP4Android.

Keywords- Entertainment, FTP4Android, mobile, remote file system, treasure hunt

1.Introduction

The mobile revolution is under everybody's eyes. The epidemic diffusion of smartphones has moved the Web and all sort of popular applications from our desks to our pockets. It has also allowed the raise of new ubiquitously available, location-based applications, no longer being driven solely by humans typing on keyboards but, increasingly, by sensors [1]. This new scenario clearly offers unprecedented potentialities for entertainment applications [2]. Yet, smartphones still suffers from some limitations such as, for instance, a limited memory for data storage. To overcome this limitation we developed FTP4Android, which is an FTP client able to manage both local and remote files. Via the FTP protocol, it is possible to establish several connections through multiple servers simultaneously. This allows the user to upload - download - delete files and create folders. The aim of the application is to provide smartphone users with a virtually infinite mass storage. This file manager presents files located on remote servers as they were available on a local directory. Users will hence be allowed to use the Internet connectivity on their smartphones to automatically retrieve files and upload updates any time a user accesses them [3], [4]. Clearly this happens at the cost of slower access time. Regarding this point, we can see in Fig. 1 a pyramidal representation of a system memory. On the top of the pyramid there are memory supports such as CPU registers that are notoriously fast but also very expensive and limited in byte capacity. Conversely, at the bottom layers we have larger and cheaper, but also slower, memory devices.

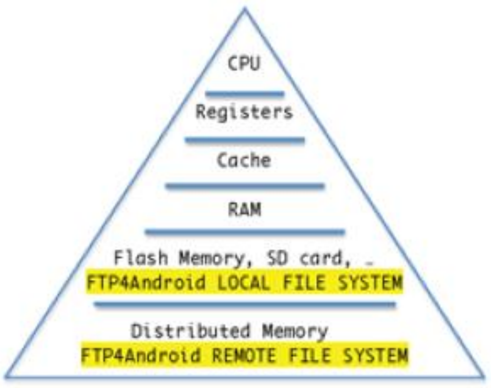

CPU
Registers
Cache
RAM
Flash Memory, SD card, ..
FTP4Android LOCAL FILE SYSTEM
Distributed Memory
FTP4Android REMOTE FILE SYSTEM

Figure 1. Memory device hierarchy

Our FTP4Android utilizes the last two layers depicted in Figure.1: a local file system on the smartphone memory and a remote file system distributed over servers in the Internet. FTP4Android extends the traditional memory hierarchy by including the layer represented by data stored on remote servers which will be characterized by the following properties.

•**Wide byte capacity.** Nowadays, a plethora of remote storage space is available on servers in the Internet.

•**Free byte cost.** It is very easy to find the aforementioned storage space for free.

•**Slow data access.** The time to access data on remote servers is even slower than that of common mass storage as it includes also network transfer time.

Finally, to demonstrate with an example how this technology could be used for entertainment purposes, we have devised a treasure hunt application for mobile users. With our application a user (the master) can save a file (e.g., a hint, a treasure map, a prize) in the remote file system. This file will be divided into several chunks that are forwarded to different remote servers. Hunters will then have to wander around the city driven by the game purpose and collecting pieces of the file (as in a puzzle) by getting close to the various servers. The rest of this paper is organized as follows. In the next section we provide a general description of FTP4Android, whereas in Section III we provide more details about its actual implementation. The pros and cons of our solution are analyzed in Section IV and a performance evaluation is reported in Section V. Section VI presents our FTP4Android-based treasure hunt application. Finally, Section VII concludes this paper.

2. FTP4Android Description

A simple description of basic functionalities implemented by FTP4Android can be provided with the help of Figure 2. In essence, files can be locally stored on the smartphone or they can be remotely stored on servers in the Internet so as to widen the memory capacity of the device. In the latter case, these files are first divided into chunks which are then sequentially uploaded on available servers. We divide files into chunks for two different goals: i) system reliability by having multiple copies of the same chunk on different servers and ii) efficiency by parallel upload/download of different chunks so as to increase the file retrieval speed. FTP4Android only requires users to install it on their smartphones (the current version is for Android operating systems).

The application is already configured to have the file system root folder in a local folder that allows writing files and folders. Moreover, it is also configured so as to employ certain remote FTP servers; however, others may be specified. General FTP servers can be used; the aim is that of allowing users to utilize their account which are generally provided for free by Internet providers. The only limitation is that chosen FTP servers must allow at least two simultaneous connections (which is generally, even if not always, true).

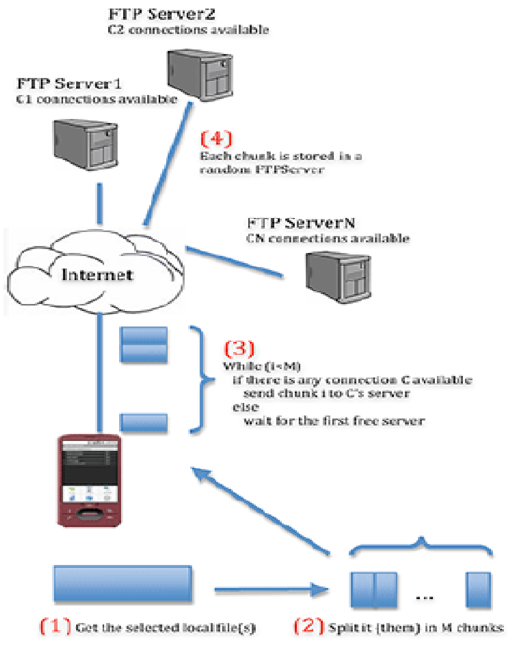

FTP Server2
C2 connections available
FTP Server1
C1 connections available
(4)
Each chunk is stored in a random FTPServer
FTP ServerN
CN connections available
Internet
(3)
While (i<M)
if there is any connection C available
send chunk i to C's server
else
wait for the first free server
...
(1) Get the selected localfile(s)
(2) Split it (them) in M chunks

Figure 2. FTP4Android general functioning

2.1. FTP Server

Before starting to illustrate the application, it is essential to understand some basic features of FTP servers. In order to interact with a FTP server, our application must know the following information:

- FTP address;
- Username and password;
- Maximum number of simultaneous connections permitted
- Timeout.

Though the meaning of the first two points is trivial, it is critic to clarify the third and the fourth ones. To avoid congestions (and for economic reasons), many FTP servers do not allow free users to transfer large amounts of data. Therefore, besides the obvious limit of storage capacity on the server, there may be other restrictions such as the maximum number of simultaneous connections allowed for the same user. Moreover there may be a maximum time of inactivity (timeout) beyond which the server does not accept anymore FTP commands from the same user.

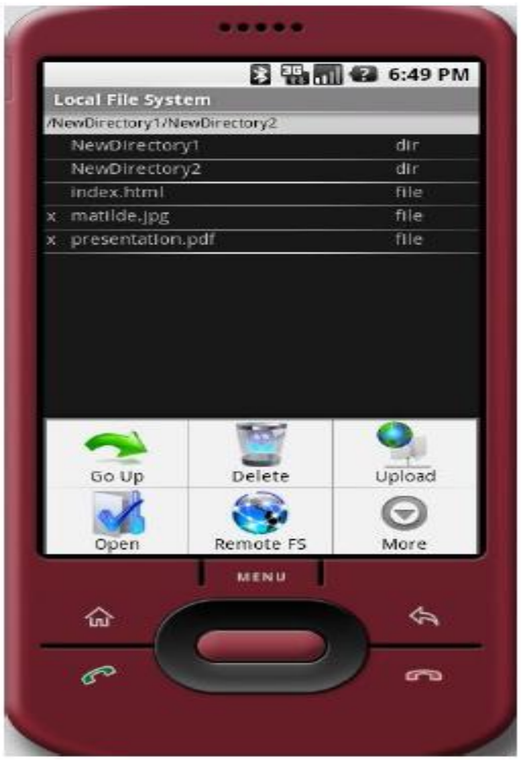
For its correct functioning, FTP4Android must be aware of these parameters and act accordingly while interacting with the chosen remote FTP servers.

2.2. The Application

FTP4Android is a software developed in Java for Google Android platform [5]. The application has a list of FTP servers and the user must have access to them. It establishes connections to the servers in order to transfer the files. For each available server,

at least two connections are established by each FTP4Android client: one is reserved for the *delete* and *create directory* operations. This reserved connection is useful to ensure that these operations can be performed quickly. The rest of the connections available are used to *upload* and *download* files. For managing and browsing the (local and remote) file system a database has been used. Through it, the application shows physical files in the *current local folder,* and displays logically associated files in the *current remote folder*. All operations available in FTP4Android can obviously be performed simultaneously, thanks to Java concurrency. These operations, that can be performed both in remote and local file system, include:
• make a directory;
• upload a file;
• download a file;
• delete a file or a directory.
A screenshots of FTP4Android running is presented in Figure 3. In the following we discuss with more details the management operations related to the remote file system.

Make a directory. In order to avoid an inconsistent remote file system, the creation of a folder is made on all FTP servers in the list of servers. In other words, this operation generates a thread for each server which will create the directory at the same path of the remote file system. Consequently, every FTP server will have the same physical file system tree.


6:49 PM
Local File System
/NewDirectory1/NewDirectory2

	NewDirectory1	dir
	NewDirectory2	dir
	index.html	file
x	matilde.jpg	file
x	presentation.pdf	file

Upload a file. By uploading a file, its chunks are created. Then, an upload thread is generated for each chunk; this thread performs the upload of the chunk on the first available connection to a server. The allocation of a chunk to a server is non-deterministic: by repeating the upload of the same file, the various chunks will probably not be assigned to the same server. If the uploaded file already existed in the remote file system then the old version is removed.

Download a file. Files are stored in chunks in the remote file system. By downloading a file, the FTP4Android application generates a download thread for each chunk and run them concurrently. In addition, a barrier is utilized to wait until all threads have completed the download of the respective chunks.

Delete a file or a directory. Starting from the file/directory selected by a user, the FTP4Android application retrieves all the information related with it (e.g., number of chunks and their location on remote servers). Then, for each FTP server, a deleting thread is generated so as to erase the interested chunks in it. In FTP4Android some of the trivial operations associated with a file system are also available:
• go to parent directory;
• open directory;
• select all/none (files and/or folders);
• switch file system (from local to remote and vice versa);
• update current folder content.

3. FTP4Android Configuration

To better understand the functioning and features of FTP4Android, we provide in the following the description of the various configuration parameters that can be modified by users.
1.Local root path. A string that specifies the absolute path of the root directory for the local file system
2.Remote root path. A string that specifies the absolute path of the root directory for the remote file system
3.Maximum number of active threads. The maximum number of threads which are allowed to run simultaneously; they can regard upload, download,
create directories, and delete files/directories.
4.Split into bytes or parts. A method for specifying either the size in bytes or the total number of chunks for each file.
5.Number of chunks. In case the file is divided into a fixed number of chunks, this parameter indicates the this number; otherwise this parameter is ignored.

6.Size of chunk. In case the file is divided into a variable number of chunks of a fixed size, this parameter indicates this size (in bytes); otherwise this parameter is ignored.

7.Create chunks in serial or parallel mode. Depending on this parameter, chunks are created in parallel (using concurrent threads) or one after the other from the beginning of the file to the end of it. In the former case, concurrent threads are governed by a barrier, which checks that all chunks have been created.

8.Reset database. If the value of this parameter is true then all the entries on the smartphone specifying which chunks are stored on which server are removed; this is used for resetting the system and deleting all files stored on remote servers.

9.New directory name. This parameter contains the default name for a new directory; an incremental counter prevents from giving an already existing name to a new directory (e.g. newDirectory1, newDirectory2, etc).

10.Server list. A list of FTP servers with the parameters for the connections (e.g., name, url, maximum number of connections allowed, etc.)

4.Advantages And Disadvantages

In this section we summarize main technical pros and cons related to dividing files into chunks to store them on remote servers.

4.1. Pros: Network Efficiency and Privacy

Through the use of threads it is possible to fully exploit the whole bandwidth available in uploading and downloading.

Example: Let's assume that we want to upload a file of 10000 Kb of size and we have the following situation:

• maximum upload bandwidth available for the device that is running FTP4Android: 100 Kb/s;
• maximum bandwidth on Server 1: 20 Kb/s;
• maximum bandwidth on Server 2: 30 Kb/s;
• maximum bandwidth on Server 3: 50 Kb/s;
• file split into 3 chunks (3334 Kb, 3334 Kb, 3332 Kb). Then, the resulting upload time will be 167 s, 111 s, and 67 s for the first, the second, and the third chunk, respectively.

Having the three threads running concurrently, the total time required to complete the upload will be 167 s, whereas with just one server (e.g. Server 3) the time required to upload in sequence the three chunks with the 50 Kb/s would be 200 s.

Furthermore, another important advantage in dividing files in chunks before forwarding them to remote servers is that in this way a user can choose different servers belonging to different providers and none of them will have all the chunks to put together the original file. In this way, users' contents will not be accessible by other people thus providing some privacy protection.

4.2. Cons: Delays and Consistency

Clearly, one of FTP4Android disadvantages relays in its use of Internet connectivity: with limited upload/download bandwidth, FTP4Android will suffer from delays in letting the user have access to its files. Moreover, if a user utilized a free server that later becomes no longer available, the visualization of the file system may become inconsistent. To prevent this problem we can introduce redundancy of chunks making the same chunk available on more servers. Yet, this would slow down the upload of a file.

5. Performance Evaluation

In this section we present two test sets we have performed on FTP4Android. The goal of these tests is to measure the performance of the application under different conditions in terms of number of chunks and active threads.

5.1. Development software

This is the development software we have utilized to implement FTP4Android:
• Eclipse "Ganymede" v. 3.4.1 build id: M20080911-1700;
• android-sdk-1.0_r2 [6];
• ftp4j library v. 1.3.1 [7];
• Java version 1.5.0_16;
• Java(TM) 2 Runtime Environment, Standard Edition
(build 1.5.0_16-b06-284);
• Java HotSpot(TM) Client VM (build 1.5.0_16-133, mixed
mode, sharing);
• Operating systems:
o Microsoft Windows Vista Business, service Pack 1;
o Mac OS X (Leopard) version 10.5.6, build: 9G55.

5.2. Test #1: Number of Chunks

First we tested the time required to upload, download, or delete a file in the remote file system, when varying the number of chunks into which the file is divided. We considered a fixed file size; thus, to a higher number of chunks correspond smaller chunk sizes. The experimental configuration is as follows:
• number of files: 1;
• size of file: 1 MB;
• number of FTP server available: 5;

• number of connection available for each FTP server: 2 (one reserved for upload/download, and the other one reserved for delete);
• maximum number of simultaneous active threads: 40 (greater than number of FTP server available);
• number of chunks: 1, 2, 4, 8, 16, and 32, respectively.

The result of this first set of experiments is reported in Figure 4. As the chart shows, the small size of the file makes preferable a small number of chunks when downloading a file from remote servers. Instead, regarding upload and delete operations, we can claim that the execution time has only a little dependency from the number of chunks.

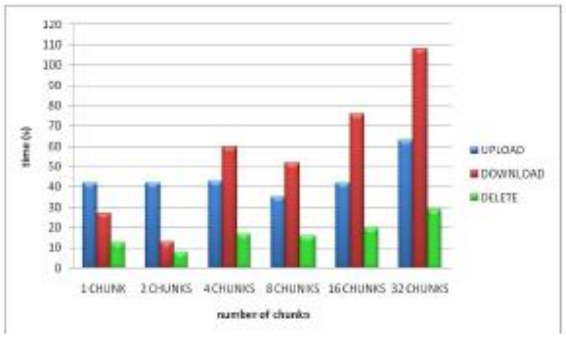


Figure 4. FTP4Android performance when varying the number of file chunks

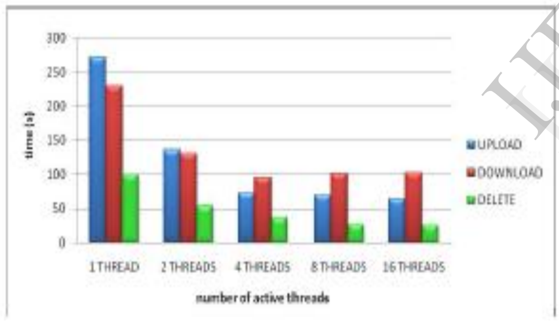


Figure 5. FTP4Android performance when varying the number of simultaneous threads for transfering file chunks

5.3. Test #2: Number of Threads

As a second set of experiments, we tested the time required to upload, download, or delete a file in the remote file system, when varying the number of active threads. Details for the experimental configuration are reported in the following list:
• number of files: 1;
• size of file: 1 MB;
• number of chunks: 32;
• number of FTP server available: 5;
• number of connection available for each FTP server: 2 (one reserved for upload/download, and the other one reserved for delete);
• number of simultaneously active threads 1, 2, 4, 8, and 16, respectively.

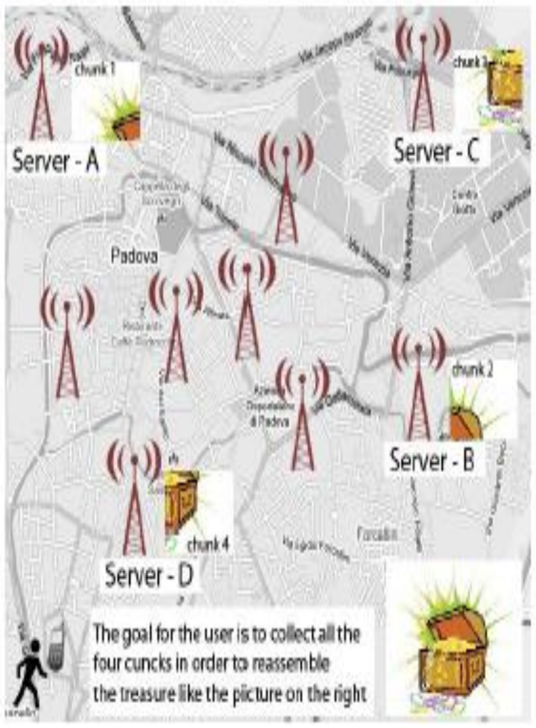
The result of the set of experiments is reported in Figure 5; in the chart it is possible to observe that when using more than just one or two threads we can reduce the transfer time.
However, not surprisingly, when the number of active threads is greater than the total number of connections available (in this case 5) the transfer time tends to stabilize and there is no
advantage in increasing further the number of active threads: the channel is already fully utilized.

The technology supporting FTP4Android can be used for many entertainment applications. The simpler one is to have available on a smartphone a huge amount of multimedia files (e.g., pictures, videos, game messages [8], [9]).

6. A Proof-Of-Concept Entertainment Application.

6.1. Treasure Hunt

Other more complex entertainment applications may be devised. Think for instance to the possibility of summarizing in a code the location of the various chunks composing a file. Then, a file sharing application could be created just by allowing users exchanging this code [10], [11]. Instead, in this work, we focused on a treasure hunt application where users have to walk to different places in a city to find information and prizes (Figure 6).



The goal for the user is to collect all the four cuncks in order to reassemble the treasure like the picture on the right

Figure 6. Treasure hunt game based on FTP4Android

Through FTP4Android a master uploads files containing information, maps, and prizes on the remote file system. Therefore, the file is automatically divided into chunks which are forwarded to different servers (e.g., Server A, B, C, and D, in Figure 6).To complete their task, users will have to walk to all the locations where servers are located so as to be able to download a file chunk (the system has been modified to allow chunk downloads by treasure hunters only with direct connection). Players will have to visit all locations where servers are physically places so as to rebuild the original file and complete the game.

The aim of this treasure hunt application could be just to participate in a game for fun, or to motivate players in visiting important cultural locations and learn historical and artistic
facts.

7. Conclusion and Future Work

The epidemic diffusion of smartphones has profoundly changed our habits, moving the Web and all our favourite applications from our PCs and notebooks to our handheld mobile devices. New ubiquitously available, location-based applications are now becoming available offering unprecedented potentialities for entertainment applications. Unfortunately, the storage capacity of these devices is limited; it may hence be useful to have the possibility to store files remotely and retrieve them when needed. To this aim, we have created FTP4Android. Our solution satisfies smartphone users' need for having infinite memory on their devices. To speed up file transfer to/from remote servers, parallel transmissions of file chunks to/from different servers are performed. We have analyzed and discussed performance when varying the number of chunks a file is divided into and the number of threads concurrently active in FTP4Android. Finally, to demonstrate how this technology could be used for entertainment purposes, we have presented a treasure hunt application for mobile users based on our FTP4Android.

References

[1] C. E. Palazzi, L. Teodori, M. Roccetti "PATH 2.0: A Participatory System for the Generation of Accessible Routes", in Proc. of the IEEE International Conference on Multimedia and Expo (ICME'10), Singapore, Jul 2010.
[2] K. Jegers, M. Wiberg, "Pervasive Gaming in the Everyday World", Pervasive Computing, IEEE, vol. 5, no. 1, pp. 78-85, Jan-Mar 2006.
[3] A. Karypidis, S. Lalis, "OmniStore: A System for Ubiquitous Personal Storage Management", in Proc. of IEEE PerCom 2006, Pisa, Italy, Mar 2006.
[4] J. Tolvanen, T. Suihko, J. Lipasti, N. Asokan, "Remote Storage for Mobile Devices", in Proc. of Comsware 2006, New Delhi, India, Aug 2006.
[5] http://www.android.com
[6]http://developer.android.com/sdk/1.0_r1/index.html
[7]http://www.sauronsoftware.it/projects/ftp4j/?lang=en
[8] G. Marfia, P. Lutterotti, S. Eidenbenz, G. Pau, M. Gerla, "Fair Multi- Media Streaming in Ad Hoc Networks through Local Congestion Control", Proc. MSWIM'08, Vancouver, Canada, Oct 2008.
[9] S. Ferretti, M. Roccetti, "Fast Delivery of Game Events with an Optimistic Synchronization Mechanism in Massive Multiplayer Online Games", Proc. ACM ACE 2005, Valencia, Spain, Jun 2005.
[10] G. Marfia, G. Pau, P. Di Rico, M. Gerla, ``P2P Streaming Systems: A Survey and Experiments", 3rd STMicroelectronics STreaming Day (STreaming Day'07), Genoa, Italy, Sep 2007.
[11] A. Karlson, G. Smith, B. Meyers, G. Robertson, M. Czerwinski, "Courier: A Collaborative Phone-Based File Exchange System", Microsoft Research Technical Report MSRTR-2008-05, 2008.