UnixWorld Online: Tutorial No. 011

Error Recovery and Restart in FTP

The explosive growth in the use of the Internet requires a re-examination of the File Transfer Protocol's ability to recover from system failures

By Raghuram Bala

Questions and comments regarding the approach outlined in this article should be directed to the author at rbala@i-2000.com.

Today, several million users access the Internet and its vast ocean of resources daily. To the layman, the Internet's most visible aspects are:

- Electronic mail
- Remote login using Telnet
- File transfer using FTP
- The World Wide Web

Many users of the Internet spend hours uploading and downloading software and data from FTP sites. This is made possible by the FTP application-level protocol of the TCP/IP suite as described by RFC 959 (144K text file). Although FTP has been around for almost two decades in various forms, not many implementations of this protocol have implemented mechanisms for recovery from system failures. Till now, this has not been a major concern because the sizes of the transferred files were relatively small (less than 1 MB) in most cases.

However, with multimedia ranging from audio to full-motion video being incorporated into entertainment, education and business software, file sizes are increasing on average. For instance, a minute long full-motion video clip could run into a megabyte or more. With technologies such as video-on-demand looming on the horizon, a lot more data transfer activity involving large files is anticipated.

Common Problems using FTP

One of the common problems that many Internet users can relate to is a system error during a file transfer. File transfer sessions get aborted as a result of:

- Server machine failure
- A failure of an intermediate host machine
- Network failure
- Client machine failure

The above reasons mainly indicate hardware failures. However, there are a number of other reasons not directly related to hardware that can abort a file transfer, including:

Heavy network load
: As more and more people get on the Information Superhighway, there is heavier loads on networks, and at times network bottlenecks that cause systems to slow down to a crawl leading to communication timeouts. A timeout occurs when one machine which is in communication with another is unable to receive an acknowledgement from the latter after a predetermined period of time. After this time window elapses, the first machine assumes that the second is unreachable.

Power outages
: If there is a fluctuation in power or a blackout, then computers without backup power supplies invariably shut down.

Software failure
: For those with Windows 3.1 software, General Protection Faults (GPF) are a daily affair. When a GPF occurs with one program, all other programs are affected. So, let us assume that you have a GPF with Microsoft Excel while you are downloading a file, then it is likely that your file transfer would be aborted in midstream.

System failures during file transfers are palatable when the file that is being transferred is small. However, it becomes annoying when a failure occurs in the midst of transferring a large file, especially when most of the transfer has taken place.

For example, let us assume that you are downloading a four megabyte file and that a system failure occurs after three megabytes have been transferred. The only recourse offered by most implementations of FTP today is for you to begin the download operation from scratch. This is an extremely painful reality, but it need not be so. In this article, I'll shed some light on the little known facts about the error recovery and restart aspects of the File Transfer Protocol.

TCP/IP in a Nutshell

The TCP/IP protocol suite forms the basis for the Internet. TCP/IP is made up of four layers:

Link
: The link layer is usually made up of the network interface card and device drivers and is primarily concerned with the physical interface.

Network
: This layer is concerned with routing of packets around a network. The most prominent of the protocols in this layer is the Internet Protocol (IP).

Transport
: This layer is concerned with the flow of data between two hosts. There are two transport protocols at this layer: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a

connection-oriented protocol and is reliable, which means it ensures that the data that flows from one host to another is delivered successfully. Often, an application would require a long message to be transmitted to another application on another machine. If the message is too large to fit in a single packet, TCP will split it up into small chunks. These packets would be routed from the source computer to the destination where they may arrive out of order. TCP on the destination machine will ensure that the packets are ordered correctly, to reconstruct the original message and present it to the Application Layer. UDP is a connectionless protocol and is unreliable, which means it does not ensure reliable delivery of packets from one host to another. The onus is on the Application layer to ensure that packets arrive reliably when using UDP.

Application

There are several applications that rely on services provided by the other layers of the TCP/IP suite. Common applications found in many implementations of TCP/IP are:

- Telnet for remote login
- FTP for file transfer
- SMTP, the Simple Mail Transfer Protocol, for electronic mail
- SNMP, the Simple Network Management Protocol

For more in-depth information on the TCP/IP Protocol Suite, refer to Reference 1.

FTP

FTP is an application-layer protocol in the TCP/IP suite, and it uses TCP as its transport-layer protocol. The primary objectives of FTP include:

- Promote sharing of files
- To shield users from variations in file systems across different platforms
- To transfer files efficiently and reliably

FTP follows the client-server model as many other TCP/IP applications do. This figure shows how this model is setup for FTP:

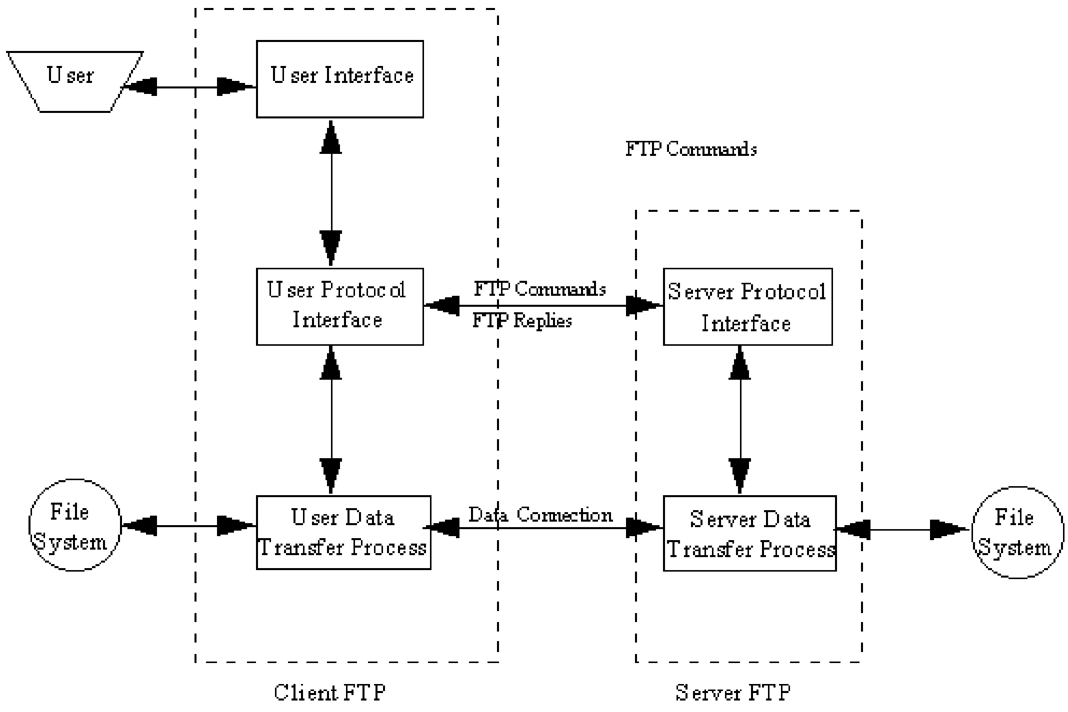

Figure 1. The FTP Model

The client half of the equation is made up of three pieces, namely, the user interface (also known as the FTP client), user protocol interpreter, and the user-data transfer function. When a user accesses a character-mode FTP client interactively, the user enters commands such as ‘‘get’’ and ‘‘put’’. Newer user interfaces are graphical, replacing these commands with graphical buttons. The commands that the user issues get interpreted by the user-protocol interpreter, which translates the request into commands understood by the FTP server. For a list of commands, refer to Reference 1. On the server end, there is a FTP server listener process (also known as a daemon) that interprets the request from the client. This connection between the user-protocol interpreter and server-protocol interface is known as a *control connection* . When a file needs to be transferred from the server to the client, a *data connection* is spawned by the client. Once data transfer is complete, the data connection is terminated. For more details, readers should refer to the References.

Users don’t need to access FTP functionality with a dedicated client. Instead, other application software can access FTP servers transparently. For example, most Web browsers, such as Netscape’s Navigator, use FTP ‘‘under the hood’’ to download files.

The way in which files are transferred and stored is determined by the following factors:

File Type
    For instance, ASCII, EBCDIC, binary
Format Control
    For instance, non-print format, Telnet format, carriage return format
Structure
    For instance, file structure, record structure

Transmission Mode
For instance, stream mode, block mode, compression mode

For more information on data representation issues, please refer to the References.

Restart and Recovery Mechanisms

The way in which error recovery and restart is detailed in RFC 959 is vague and implementation details are not mentioned. The primary mechanism is use of a restart marker that is only available when using block or compressed transmission mode. With block transfers, a file is transferred in chunks made up of a header portion followed by a data portion. The header portion has a descriptor and a byte count for the data portion. The one-byte descriptor field describes the data block. Certain bits are set for a special meaning. For instance, if the most significant bit is set to one, it means that the data block marks the end of a record. In that vein, if the fourth most significant bit is enabled, then it indicates that the data block holds a restart marker.

In compressed-mode transfers, restart markers are preceded by an escape sequence that is a double byte. The first byte is all zeroes and the second is a descriptor byte similar to that used in block-transfer mode.

What is a restart marker and how is it going to help us in recovering from a system failure? Restart markers (also known as *checkpoints*) are milestones during a file transfer process. Should a failure occur, the file transfer need not be restarted from the beginning, and instead could proceed from the last recorded milestone.

Readers should note that in order for any error recovery as specified by RFC 959 to be implemented effectively, it requires cooperation among all implementors of FTP client and server programs to agree on a common format for restart markers.

Proposal for a Better Restart Marker

Let us assume that an FTP client and an FTP server support a common recovery and restart scheme. Now, suppose the FTP client wants to download a four-megabyte file from the server. The server may decide to embed a restart marker every 100K bytes, say. Then, if a system failure occurs after transferring 3,213,517 bytes, say, the file transfer process could be rolled back and started from the 3,200,000 byte mark. Is this good enough? Well in most cases the answer would be ‘‘yes’’. What if the file that was being transferred is modified before the FTP client decides to rollback and continue to download the remainder of the file? In this case, there is no guarantee that the file that was transferred would be coherent to the intended audience because it would essentially be a mish-mash of two files.

Hence, let me now propose a standardized restart marker that would solve this problem. A simple solution would be to store the file size of the file to be downloaded in the restart marker together with a byte count indicating the cumulative number of bytes downloaded thus far. When a failure occurs, the file size from the restart marker can be compared with the file size at the time of error recovery to see if they match. If they match, then the file transfer can proceed, otherwise, the FTP client is notified that the file has been modified and that recovery is not possible.

There is an inherent flaw in the above solution. Files can change without file sizes having to change! So, file size is not a reliable gauge for determining whether a file has been modified or not. Instead a better measure would be a *time stamp*. This time stamp would include the date and time when a file was last modified. Our proposal for a restart marker will consist of a byte-count followed by a time stamp:

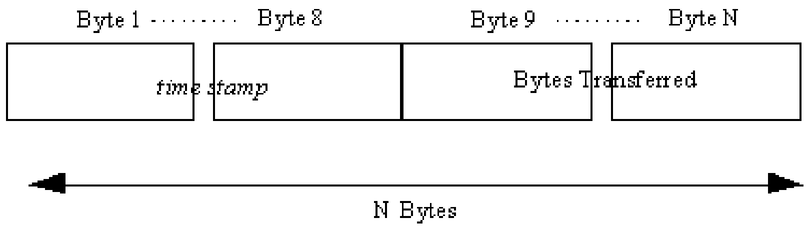

Figure 2. Proposed Restart Marker

The proposed restart marker consists of *N* bytes, where *N* is an integer greater than or equal to nine, and the first eight bytes store the time stamp for the last- modified time of the file being transferred. The nineth to the *N*th byte stores the file size. The value assigned *N* is based on the number of bytes required to store the file size. For example, if the file size is 50 bytes long, then *N* would be 8 + 1 = 9. If the file size is one gigabyte, then 8 + 30 = 38 is employed

Example

In this section, I shall go through the time line for an FTP download procedure which has a system failure and subsequent recovery. This figure shows a time line:

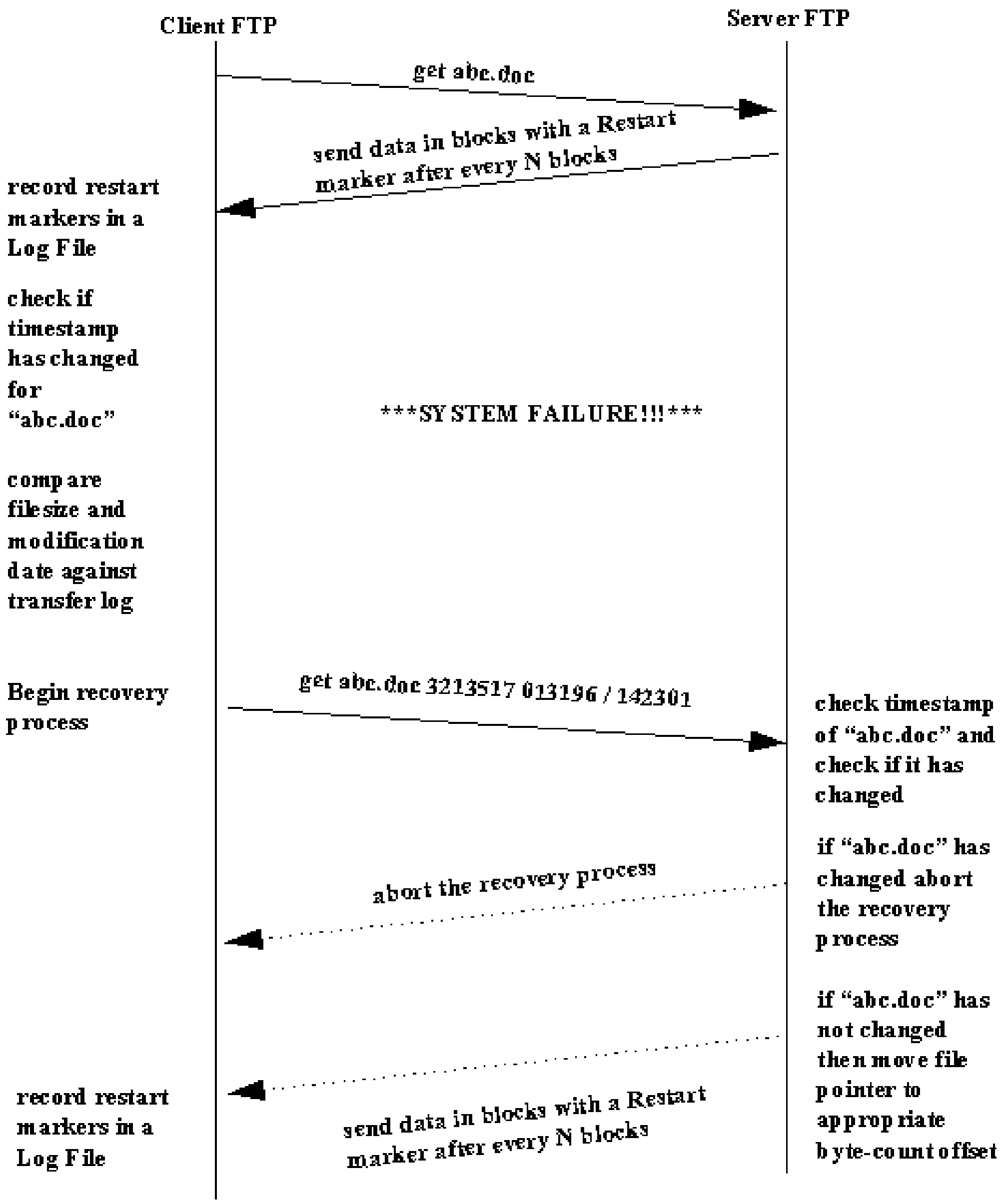

Figure 3. Timeline for Restart/Recovery

The events that take place during the file transfer process are in the following chronological order:

1. FTP client issues download request, for instance, **`get abc.doc`**
2. FTP server receives download request and begins downloading `abc.doc`. Every 100K bytes, it inserts a restart marker with a byte-count and time stamp.
3. FTP client receives data blocks and creates a local version of `abc.doc`. Whenever it comes across a restart marker, it updates a transfer log as to how many bytes have been transferred and remote file's time stamp. In addition, the transfer log would contain the local file's time stamp. Assuming the FTP server does not have an exclusive lock on `abc.doc`, it is possible that `abc.doc` is modified even when no system failure takes place. Hence, the two successive time stamps can be compared by the FTP client to ensure that there is no loss of data integrity during the file transfer. If time

stamps don't match, abort transfer and inform FTP server. Otherwise continue.

4. System failure occurs!!
5. FTP client reads its transfer log and extracts the local file's time stamp and byte count. Comparison is made between bytes transferred from server and local file size, and the time stamp from the transfer log with the local file's last modification date. This is to ensure that no modifications have been made to `abc.doc` locally. If there is a mismatch, do not proceed with error recovery.
6. FTP client issues request to FTP server to restart download passing restart marker that contains byte-count and time stamp for instance, **`get abc.doc 3213517 013196 / 142301`**
7. FTP server receives restart request and compares the time stamp with server copy of `abc.doc`. If time stamps match, then it moves file pointer to an offset equivalent to the byte count and continues to download from that point.

Note that a transfer Log is maintained on the client end in the scheme shown above. This transfer log may be implemented as a simple file whose records have the following structure:

```
struct {
    char* filename;         // should include path (if any)
    long  bytestransferred; // bytes transferred
    TIMESTAMP rt;           // last server file
                            // modification time stamp
    TIMESTAMP ct;           // last client file
                            // modification time stamp
} LOGSTRUCT;
```

Algorithms

Listing 1A presents some pseudo-code for implementing the FTP protocol discussed above in the client and Listing 1B for the server. These algorithms are presented at a high-level and interested readers should refer to Reference 4 for more details. All functions starting with the prefix ''svr'' are server functions and would be called from the client via RPCs. But I have omitted details regarding RPCs here.

Conclusion

It is apparent that error recovery and restart are essential in implementations of the File Transfer Protocol. However, it requires cooperation among software vendors and the industry in general to bring about a consensus opinion on the format of a restart marker. In this article, I have proposed a format for a restart marker that I believe helps in furthering the cause of improvements to FTP.

References

1. Stevens, W. Richard. *TCP/IP Illustrated, Volume 1. The Protocols*. Reading, Mass: Addison-Wesley. ISBN: 0-201-63346-9

2. Comer, Douglas E. *Internetworking With TCP/IP, Volume 1: Principles, Protocols, and Architecture*. Englewood Cliffs, N.J.: Prentice-Hall. ISBN: 0-13-468505-9

3. Official FTP protocol specification in RFC 959 (144K text file) (ftp://ds.internic.net/rfc/rfc959.txt).

4. Stevens, W. Richard. *Unix Network Programming*. Englewood Cliff, N.J.: Prentice-Hall Software Series. ISBN: 0-13-949876-1

5. Stallings, William. *Data and Computer Communications, Third Edition*. MacMillan Publishing Company, New York, N.Y., ISBN: 0-02-415454-7

Copyright © 1995 The McGraw-Hill Companies, Inc. All Rights Reserved.
Edited by Becca Thomas / Online Editor / UnixWorld Online / editor@unixworld.com

Last Modified: Wednesday, 21-Feb-96 08:50:40 PST