Network Working Group
Request for Comments: 1068

A. DeSchon
R. Braden
ISI
August 1988

Background File Transfer Program (BFTP)

Status of This Memo

This memo describes an Internet background file transfer service that is built upon the third-party transfer model of FTP. No new protocols are involved. The purpose of this memo is to stimulate discussion on new Internet service modes. Distribution of this memo is unlimited.

1. Introduction

For a variety of reasons, file transfer in the Internet has generally been implemented as an interactive or "foreground" service. That is, a user runs the appropriate local FTP user interface program as an interactive command and requests a file transfer to occur in real time. If the transfer should fail to complete for any reason, the user must reissue the transfer request. Foreground file transfer is relatively simple to implement -- no subtleties of queuing or stable storage -- and in the early days of networking it provided excellent service, because the Internet/ARPANET was lightly loaded and reasonably reliable.

More recently, the Internet has become increasingly subject to congestion and long delays, particularly during times of peak usage. In addition, as more of the world becomes interconnected, planned and unplanned outages of hosts, gateways, and networks sometimes make it difficult for users to successfully transfer files in foreground.

Performing file transfer asynchronously (i.e., in "background"), provides a solution to some of these problems, by eliminating the requirement for a human user to be directly involved at the time that a file transfer takes place. A background file transfer service requires two components: a user interface program to collect the parameters describing the required transfer(s), and a file transfer control (FTC) daemon to carry them out.

Background file transfer has a number of potential advantages for a user:

- No Waiting

  The user can request a large transfer and ignore it until a notification message arrives through some common channel (e.g., electronic mail).

- End-to-end Reliability

  The FTC daemon can try a transfer repeatedly until it either succeeds or fails permanently. This provides reliable end-to-end delivery of a file, in spite of the source or destination host being down or poor Internet connectivity during some time period.

- Multiple File Delivery

  In order for background file transfer to be accepted in the Internet, it may have to include some "value-added" services. One such service would be an implementation of a multiple file transfer capability for all hosts. Such a facility is suggested in RFC-959 (see the description of "NLST") and implemented in some User-FTP programs.

- Deferred Delivery

  The user may wish to defer a large transfer until an off-peak period. This may become important when parts of the Internet adopt accounting and traffic-based cost-recovery mechanisms.

There is a serious human-engineering problem with background file transfer: if the user makes a mistake in entering parameters, this mistake may not become apparent until much later. This can be the cause of severe user frustration. To avoid this problem, the user interface program ought to verify the correctness of as many of the parameters as possible when they are entered. Of course, such foreground verification of parameters is not possible if the remote host to which the parameters apply is currently unreachable.

To explore the usefulness of background file transfer in the present Internet, we have implemented a file-mover service which we call the Background File Transfer Program or BFTP.

Section 2 describes BFTP and Section 3 presents our experience and conclusions. The appendices contain detailed information about the

user interface language for BFTP, a description of the program organization, and sample execution scripts.

2. Background File Transfer Program

2.1 General Model

In the present BFTP design, its user interface program and its FTC daemon program must execute on the same host, which we call the BFTP control host.

Through the user interface program, a BFTP user will supply all of the parameters needed to transfer a file from source host S to destination host D, where S and D may be different from the BFTP control host. These parameters include:

- S and D host names,
- login names and passwords on S and D hosts, and
- S and D file names (and optionally, directories).

The user may also specify a number of optional control parameters:

* Source file disposition -- Copy, move (i.e., copy and delete), or simply delete the source file. The default is copy.
* Destination file operation -- Create/Replace, append to, or create a unique destination file. The default is create/replace ("STOR").
* FTP Parameters -- Explicitly set any of the FTP type, mode, or structure parameters at S and D hosts.
* Multiple Transfers -- Enable "wildcard" matching to perform multiple transfers.
* Start Time -- Set the time of day for the first attempt of the transfer. The default is "now" (i.e., make the first attempt as soon as the request has been queued for the FTC daemon).

Finally, the user specifies a mailbox to which a completion notification message will be sent, and "submits" the request to the FTC daemon queue. The user can then exit the BFTP user

interface program.

If the transfer should fail permanently, the FTC daemon will send a notification message to the user's mailbox. In the event of a temporary failure (e.g., a broken TCP connection), the FTC daemon will log the failure and retry the transfer after some timeout period. The retry cycles will be repeated until the transfer succeeds or until some maximum number of tries specified has been reached. In either case, a notification message will then be sent to the user's mailbox.

The user can check on the progress of the transfer by reentering the BFTP user interface program, supplying a key that was defined with the request, and displaying the current status of the request. The user may then cancel the request or leave it in the queue.

The BFTP program includes a server-Telnet module, so it can be executed as a remotely-accessible service that can be reached via a Telnet connection to the BFTP well-known port (152). This allows a user on any Internet host to perform background file transfers without running BFTP locally, but instead opening a Telnet connection to port 152 on a BFTP service host. Of course, a user can also run the local BFTP user interface program directly on any host that supports it and for which the user has login privileges.

The next section discusses how BFTP uses standard FTP servers to perform the transfers, while the following section covers the user interface of BFTP.

2.2 File Transfer Mechanics for BFTP

The BFTP makes use of the "third party" or "Server-Server" model incorporated in the Internet File Transfer Protocol [RFC-959]. Thus, the FTC daemon opens FTP control connections to the existing FTP servers on source host S and destination host D and instructs them to transfer the desired file(s) from S to D. The S and D hosts may be any two Internet hosts supporting FTP servers (but at least one of them must support the FTP "PASV" command). This approach allows the implementation of a background file transfer capability for the entire Internet at a very low cost.

Figure 1 illustrates the BFTP model of operation. Note that the BFTP control host is not necessarily the same as S or D. Figure 2 illustrates the FTP command interchange used in a typical Server-Server file transfer operation; this may be compared with the User-Server FTP scenario illustrated in Section 7 of RFC-959.

Since BFTP may be asked to transfer files between any two hosts in the Internet, it must support all the file types and transfer modes that are defined in RFC-959, not just a subset implemented by particular hosts.

BFTP supports the transfer of a set of files in a single request, using the standard technique:

(1) Send an NLST command to the source host S, specifying a pathname containing "wildcard" characters. The reply will contain a list of matching source file names.

(2) Execute a separate transfer operation for each file in this list. The destination file name in each case is assumed to be the same as the source file name; this requires that these names be compatible with the naming conventions of D.

It will typically be necessary to specify working directories for the transfers at S and D, so the file names will be simple, unstructured names on each system.

This approach depends upon the wildcard matching capability of the source host S. A more general implementation would acquire a complete list of the file names from the source host and do the matching in the FTC daemon, for example using a regular-expression matcher. Another useful extension would be a general pattern-matching file name transformation capability (e.g., like the one included in the 4.3BSD version of FTP) to generate appropriate destination pathnames for multiple requests.

Figure 1 -- BFTP Model of Operation

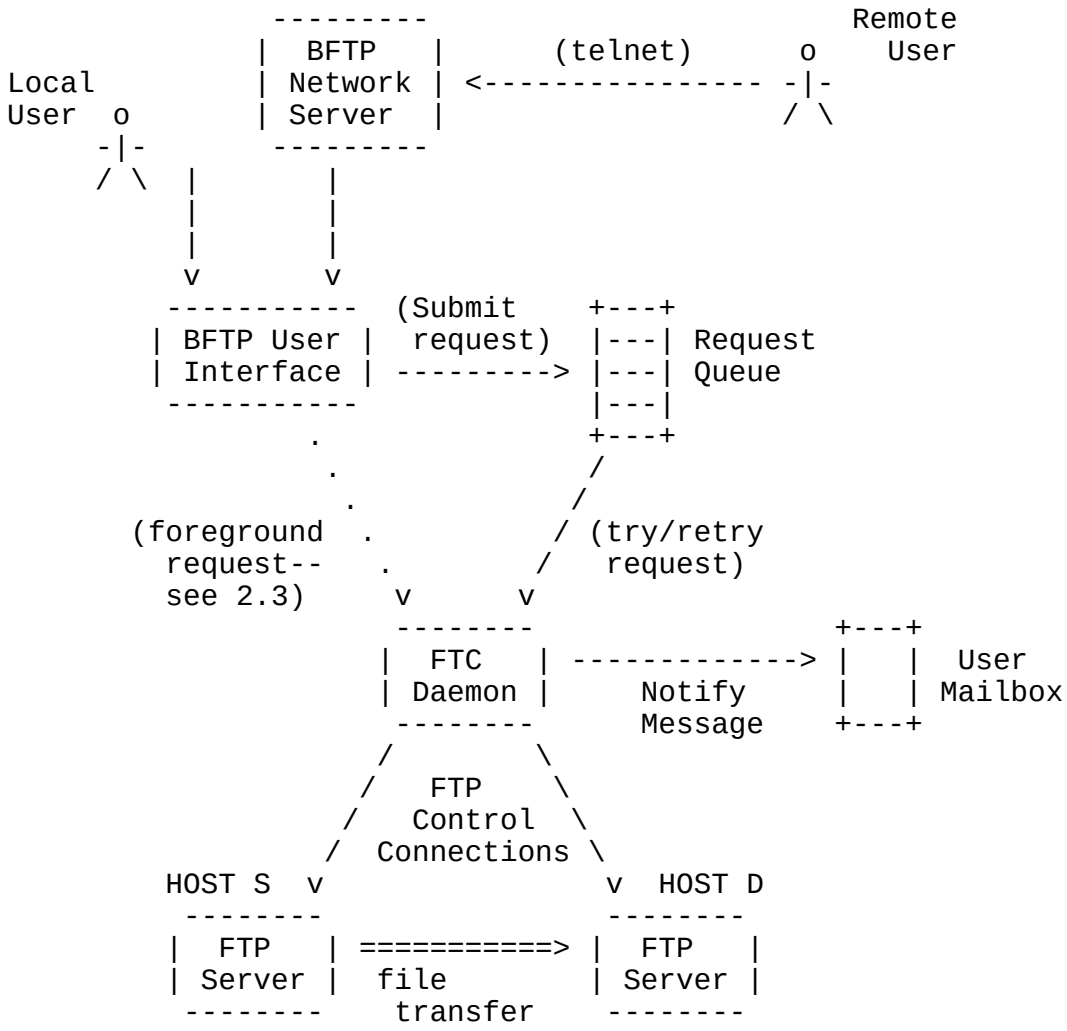
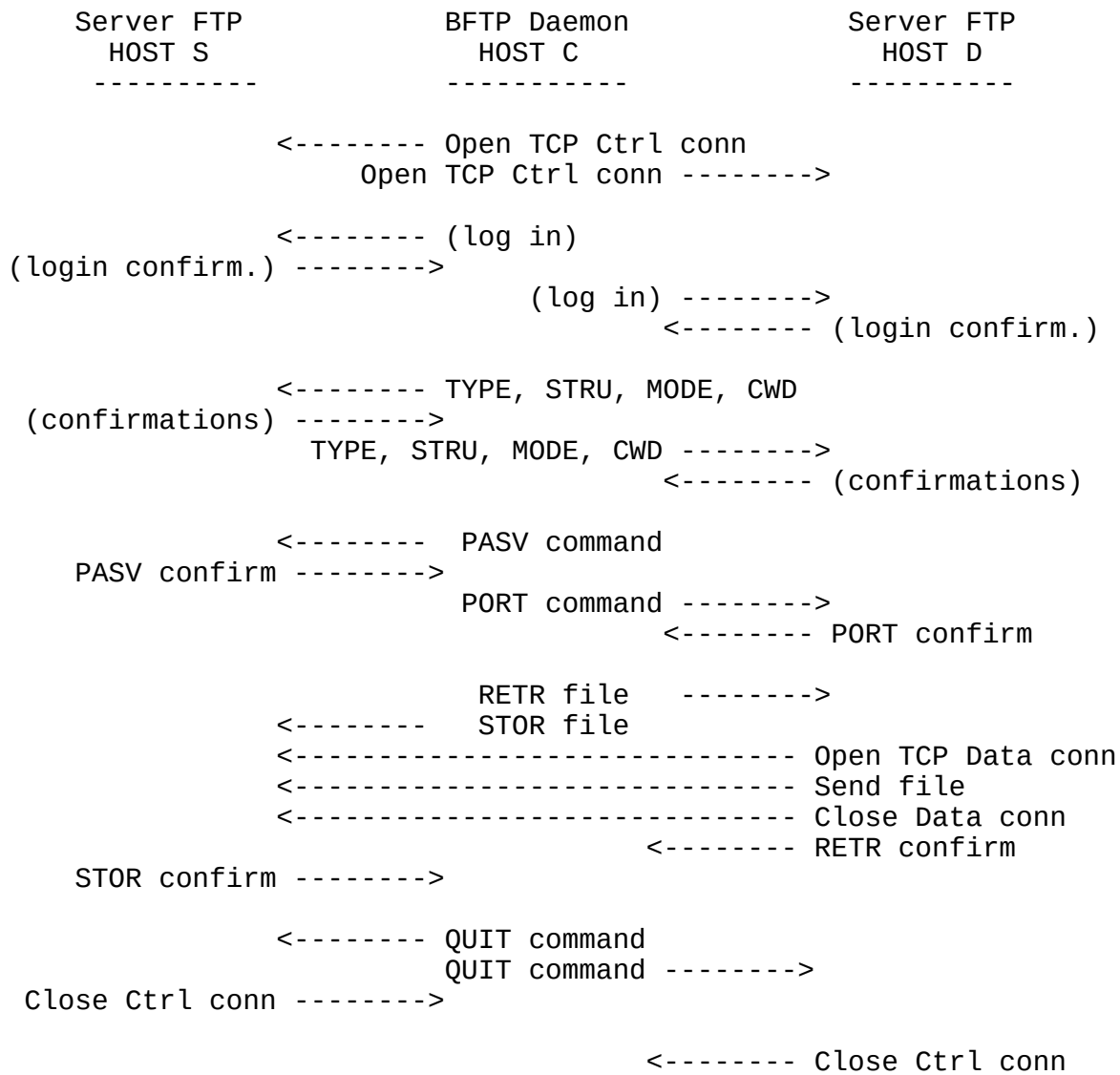


Figure 2 -- Server-Server File Transfer



BFTP currently utilizes the following Server-FTP commands [RFC-959]: USER, PASS, ACCT, PASV, PORT, RETR, STOR, STOU, CWD, NLST, MODE, STRU, TYPE, and QUIT.

The FTC daemon attempts to work around FTP servers that fail to support certain commands. For example, if a server does not support the optional command "CWD", the FTC daemon will attempt to construct a complete path name using the source directory name and the source file name. However, it is necessary that at least one of the two hosts support the FTP passive (PASV) command. While many FTP server implementations support do this command, some (in particular, the 4.2BSD FTP) do not. The PASV command was officially listed as being optional in RFC-959.

2.3 Reliable Delivery

The reliable delivery function of BFTP is analogous to reliable delivery in a transport protocol like TCP. Both depend upon repeated delivery attempts until success is achieved, and in both cases the choice of the retry interval requires some care to balance overhead against unresponsiveness.

Humans are impatient, but even their impatience has a limit. If the file cannot be transferred "soon", a human will turn to another project; typically, there is a tendency for the transfer to become less urgent the longer the wait. The FTC daemon of BFTP therefore starts each transfer request with a very short retry interval -- e.g., 10 minutes -- and then doubles this interval for successive retries, until a maximum interval -- e.g., 4 hours -- is reached. This is essentially the exponential backoff algorithm of the Ethernet, which is also used by transport protocols such as TCP, although BFTP and TCP have quite different rationales for the algorithm.

We must also define the meaning of reliable transmission for a multiple-transfer request. For example, the set of files selected by wildcard characters in a pathname is not well defined; the set may change while the request is pending, as files are created and deleted. Furthermore, it is unreasonable to regard the entire multiple transfer as a single atomic operation. Suppose that transferring a set of files fails part way through; for an atomic operation, the files which had been successfully transferred would have to be deleted pending the next retry of the entire set. This would be ridiculously inefficient and may be impossible (since the communication path may be broken when it is time to issue the deletion requests).

BFTP addresses these issues in the following manner:

* For a multiple file operation, the FTC daemon saves the file name list returned by the first successful NLST command in the request queue entry. This name list determines the set of source files for the transfer; there can be no later additions to the set.

* The FTC daemon maintains a transfer status pointer. On each retry cycle, it tries to transfer only those files that have not already been successfully transferred.

* The request is complete when all the individual file transfers have been successful, a permanent failure has occured, or when the retry limit is reached.

* The notification message to the user lists the status of each of the multiple files.

2.4 BFTP User Interface

The purpose of BFTP is to simplify the file transfer process and to place the burden of reliability on the BFTP control host. We have attempted to provide a "user friendly" command interface to BFTP, similar in flavor to the user interface of the TOPS-20 operating system. This interface provides extensive prompting, defaulting, and help facilities for every command.

For a list of all BFTP commands, the user may enter "?<Return>" at the main BFTP prompt ("BFTP>"). Entering "help<Return>" and "explain<Return>" will provide increasing levels of explanatory material. To obtain information on a particular command, "help <command name><Return>" may be entered. The 'quit' or 'exit' command will exit from BFTP. Command and subcommand names may be abbreviated to the shortest unique sequence for that context; alternatively, a partial name can be automatically completed by typing <Return>.

The normal procedure for a BFTP user is to set up a set of parameters defining the desired transfer and then submit the request to the FTC daemon. To give the user the maximum flexibility, BFTP supports three modes of submission:

o Background Operation

  To request a reliable background file transfer, the user will issue the BFTP 'submit' command to the FTC daemon.

- Foreground Verification, Background Operation

  The BFTP 'verify' command may be used to ascertain that file transfer parameters are valid. It causes BFTP to connect to the FTP servers on both the source and the destination hosts (if possible), log into both, verify the FTP parameters, and verify that the specified source file is present.

  Once the 'verify' command has successfully completed, the user can issue the 'submit' command to schedule the actual file transfer.

- Foreground Operation

  The BFTP 'transfer' command will perform the specified third-party transfer in foreground mode. This is illustrated by the dotted path bypassing the queue in Figure 1.

The easiest way to set up the parameters is to issue the 'prompt' command, which will prompt the user for all of the basic parameters required for most transfers. Certain unusual parameters must be set with the 'set' command (see Appendix B for details).

When entering any parameter, the following control characters may be used:

? will display help text for the parameter, indicating its meaning, the choices, and the default, and then reprompt for the parameter.

<ESC> will display the default value (or the last value set) for this parameter. The user can accept this default by entering <Return>, or else erase it with Control-W and enter a different value for the parameter, followed by <Return> to accept the entered value.

<Control-W>
will erase the value typed or displayed for current parameter.

<Return>
will accept the value displayed for this parameter, and continue to the next parameter, if any. If the user has not typed a value or used <ESC> to display the default, <Return> will display the default and then accept it.

It is important to provide a means for a user to obtain status information about an earlier request or even to cancel an earlier request. However, these functions, especially cancellation, must be controlled by some user authentication. We did not want to build a user authentication database with each BFTP instance or require login to BFTP itself, and there is no Internet-wide user authentication mechanism. We adopted the following weak authentication mechanism as a compromise:

* When the 'submit' command is issued, it prompts the user for a character string called a "keyword", which recorded with the request.

* This keyword can be entered later as the argument to a 'find' command, which will display the status of all requests with matching keywords.

* Similarly, the keyword may be used to cancel the corresponding request.

If two different users happen to choose the same keywords, of course, this scheme will not protect each other's requests from accidental or malicious cancellation. However, a notification message will be sent at the time that a cancellation occurs.

To make a series of similar requests, the user needs only to change the individual parameters that differ from the preceding request and then issue a new 'submit' command, for each request. There are commands for individually setting each of the parameters that 'prompt' sets -- and 'time' -- to provide a shortcut for BFTP experts. A simpler but lengthier procedure is to use the 'prompt' command to run through the current set of parameters, reentering the parameters that must change and using the sequence <ESC><return> to retain the previous value for each of the others. The same procedures may be used to correct a mistake made in entering a particular parameter.

The current settings of all the BFTP parameters can be displayed at any time with the 'status' command, while the 'clear' command will return all parameters to their initial values. Finally, the 'request' command allows the user to save the current set of parameters in a file or to restore the parameters from a previously-saved file.

There is also a window-based BFTP user interface for use on a Sun Workstation, described in Appendix A. The complete list of BFTP commands is presented in Appendix B.

3. Experience and Conclusions

BFTP has been available to users at ISI for some months. Users have reported a number of advantages of using BFTP:

(a) Some users prefer the prompting style of BFTP to the user interface of the foreground FTP they normally use.

(b) The BFTP "verify" command allows the user to verify that host names, passwords, and filenames are correct without having to wait for the entire transfer to take place.

(c) Since results are returned through the mail system, a transfer can occur without tying up a terminal line, a phone line, or even a window.

BFTP must be able to communicate with a variety of Server-FTP implementations, and we have observed much variation in the commands supported, error handling, and the timing in these servers. Some of the problems we have encountered are:

(1) Some systems (e.g., 4.2BSD) do not support the PASV command.

(2) 4.2/3BSD systems return a non-standard response to the NLST command. Instead of returning a list of complete path-names, they use an ad hoc format consisting of a directory name followed by a list of files.

(3) 4.2/3BSD systems may return a "permanent negative completion reply" (a 5xx FTP reply code) as a result of a communications failure such as a broken TCP connection. According to RFC-959, the appropriate response is a "transient negative completion reply" (a 4xx FTP reply code), which would inform the BFTP that the transfer should be retried.

(4) A number of servers return badly formatted responses. An example of this is the 4.2/3BSD response to an NLST command for a non-existent file name: an error string which is not preceded by a numerical response code.

To diagnose problems that do occur, we have found it very useful to have a complete record of the interchange between the FTC daemon and the two FTP servers. This record is saved and is currently always included in the notification message mailed to the user (see Appendix D for an example). As we get more experience with this program, some of the details of the transfer may be omitted from this log.

The use of library routines shared between modules makes it relatively easy to implement additional user interface programs. We are currently experimenting with a window version of BFTP, the "bftptool", which runs in the SunView environment, and is described in Appendix A. Some additional interfaces that might be useful are:

- A command line interface for use in shell scripts and "Makefiles".
- A more general library interface which would make it easy to invoke BFTP from a variety of programs.
- Additional full-screen form based interfaces, for example a tool running in X-Window system environment.

Lastly, BFTP would benefit from the resolution of the following open protocol issues:

- There currently exist no provisions for Internet-wide user authentication. In the BFTP context, this means that passwords required for a file transfer must be present in BFTP request files. The security of these passwords is subject to the limitations of the file system security on the BFTP control host. Anonymous file transfer provides a partial solution, but a more general, long term solution is needed.
- Better mechanisms are needed to cope with the diversity of real file systems in the Internet.

  For example, an extension could be made to the FTP protocol to allow the daemon to learn the delimiter conventions of each host file system. This could allow a more flexible and powerful multiple-file facility in BFTP. This could include the automatic transfer of directory subtrees, for example.

4. References

[RFC-959] Postel, J., and J. Reynolds, "File Transfer Protocol (FTP)", RFC-959, USC/Information Sciences Institute, October 1985.

Appendix A -- BFTP Implementation Structure

BFTP has been implemented on both a Sun workstation running Sun OS 3.4 (based on 4.2BSD) and a VAX running 4.3BSD. The program modules are: the local user interface programs "bftp", the Internet server program "bftpd", and the FTC daemon "fts". BFTP makes use of the "at" command, a UNIX batch job facility, to submit requests and execute the daemon. An additional user interface program, the "bftptool", is available for Sun OS 3.4, and runs in the SunView environment.

BFTP keeps its state in a set of control files: request files, command files, and message files. These files are stored in the home directory specified for the environment of the process running "bftp". If a user is running "bftp" directly, this will typically be the user's home directory. In the case where a user has made a Telnet connection to the well-known port 152 on a BFTP service host, "bftp" is started by "bftpd" (or "inetd", indirectly). As a result, the control files will be owned by the user-id under which "inetd" was started, normally "root", and stored in the top level directory "/". Note, however, that under BFTP all user files are written by the FTP servers, which are presumed to enforce the operating systems' access control conventions. Hence, BFTP does not constitute a system integrity exposure.

A.1 User Interface Program

The BFTP user interface program "bftp" may be run directly via a UNIX shell. Once the program has been started, the prompt "BFTP>" will appear and commands may be entered. These commands are described in detail in Appendix B.

A.2 Tool-Style User Interface Program

The BFTP user interface program "bftptool" may be started from a shell window in the SunView environment on a Sun workstation. The BFTP commands may be selected via the left mouse button. The various file transfer parameters appear in a form-style interface; defaults and multiple-choice style parameter values can be filled in via menus. An advantage of this form-style interface program is that it is possible to view all of the file transfer parameters simultaneously, providing the user with a sense for which parameter values might be mutually exclusive.

Help information can be displayed in a text subwindow by positioning the on-screen mouse pointer over a command or a parameter, and clicking the center mouse button. (No standard mechanism for displaying help information is currently included in

the SunView package.)

The commands used in the "bftptool" are for the most part very similar to the commands described in Appendix B. Request submittal and the execution of the FTC daemon are identical for the "bftp" and the "bftptool" interface programs.

A.3 Internet Server

The Internet server program "bftpd" can be invoked by opening a Telnet connection to a well-known port, and does not require login. The "bftpd" program runs under "inetd", the standard BSD4.x well-known port dispatcher. When a SYN arrives for the BFTP well-known port, "bftpd" opens the TCP connection and performs Telnet negotiations. It then passes control to the user interface "bftp" which allows the user to enter file transfer requests.

A.4 BFTP Server Daemon

The BFTP file transfer control daemon program is named "fts" (for "File Transfer Service"). This module contains code to actually cause a single file transfer operation using the FTP server-server model as shown in Figures 1 and 2. It is invoked with the command "fts <request-file>". The <request-file> contains the necessary parameters for the file transfer, in ASCII format, separated by linefeeds. Such a request file may be created by the user interface program, "bftp".

As a byproduct of the development of BFTP, "fts" represents a server-server FTP driver that can be run independent of the "bftp" program. Parameters used in the file transfer are read from a request file, which is created and accessed via library routines which can be shared between modules. This could be used to perform FTP’s under program control.

Appendix B: BFTP Command Summary

B.1 Special Editing Characters

In the "bftp" program, the special editing characters for command words, subcommands, and parameter fields are as follows:

<return>	Accept current command/field.
<escape>	Complete current command/field, or display default.
<space>	Complete and delimit current command.
<delete>	Erase last character.
control-L	Refresh screen.
control-R	Refresh line.
control-U	Erase line.
control-W	Erase current token.
?	List legal options.

B.2 BFTP Commands

The remainder of Appendix B consists of a list of the BFTP commands. Each command should be followed by a carriage-return. In the description of the syntax for each command, square brackets "[]" are used to indicate a ssubcommand, or a list of possible subcommands, which are separated by the "|" character. Angle brackets "<>" are used to indicate a description of a parameter where the choices would be too numerous to list, for example "<host name/number>".

B.2.1 Clear Command

Return all parameters to their default values.

```
clear
```

B.2.2 Destination Commands

Set the destination directory.

```
ddir <directory name>
```

Set the destination file name.

```
dfile <file name>
```

Set the destination host, user, and password.

```
dhost <host name/number> <login> <password>
```

B.2.3 Explain Command

Display a short explanation of how to use BFTP.

```
explain
```

B.2.4 Find Command

Find and display a previous request.

```
find
```

BFTP will prompt for the request id, which is printed when the request is first submitted. An example of a request id is "bftp583101774". BFTP also prompts for the request keyword, which was determined by the user when the request was first submitted. If no keyword was specified, a <CR> should be typed. If no request id is entered, BFTP will display all requests which contain a matching keyword.

```
RequestID (optional): <bftp-request-id>
RequestKeyword: <keyword>
```

After BFTP has displayed a summary of a matching request, it asks whether the request is to be changed, or canceled.

```
Do you wish to change this request? [yes | no]
Do you wish to cancel this request? [yes | no]
```

If the user indicates that the request is to be changed, BFTP will read in the parameters and cancel the existing request. At this point the user may make any desired changes and use the "submit" command to requeue the request. At this point a new request id will be assigned and displayed.

Although this may happen extremely rarely, if at all, it is possible that a system crash (or the interruption of the BFTP program) at a particularly inopportune moment may leave a request which is not queued. When the "find" command locates such a request, it displays the warning:

```
Your request is NOT currently queued.
```

If this happens, the request may be read in and resubmitted using the following procedure:

```
Your request is NOT currently queued.
Do you wish to change this request? yes

  (BFTP displays the parameters that have been read in.)

Previous request canceled.
Use the 'submit' command to submit a new request.
```

B.2.5 Help Command

Print local help information.

```
help
help <command>
```

B.2.6 Quit Command

Clear parameters and exit the BFTP program.

```
quit
```

B.2.7 Prompt Command

Prompt for commonly-used parameters.

```
prompt
```

The following are the parameters that BFTP prompts for:

```
      copy/move/delete: [copy | move | delete]
      ascii/ebcdic/image/local:
            [ascii|ebcdic] [nonprint|telnet|carriage-control]
or
            [image]
or
            [local] <byte size>
(see "set type" for additional information)

      Source --
          Host: <host name/number>
          User: <login>
          Password: <password>
          Dir: <directory including a delimiter, e.g., "/" or ">">
               (either an absolute path, or relative to the login)
          File: <file name>

      Destination --
          Host: <host name/number>
          User: <login>
          Password: <password>
          Dir: <directory>
          File: <file name>
```

Once the prompting has been completed, the current values of all parameters will be displayed. Parameters not mentioned in the prompting will be initialized with default values, and may be changed via the "set" commands.

B.2.8 Request Commands

The request commands enable the user to save a set of BFTP parameters in a "request-file" for future use. Subcommands are provided to to list all available request-files, or to read, write, or delete a request-file. All request-files are stored in the user's home directory. Therefore, this facility is not available when the user is accessing BFTP by telneting to port 152.

Delete request file "bftp-save.name".

```
request delete <name>
```

List all bftp-save files.

```
request list
```

Read a request file in as the current request.

```
request load <name>
```

Save the current request in a file named "bftp-save.name".

```
request store <name>
```

B.2.9 Set Commands

The "set" commands have complex subcommand structures and are used to set many of the less commonly used FTP parameters. The subcommands of "set" are as follows:

Set the account for the source/destination login.

```
set account [source | destination] <account string>
```

Set to true to append to destination file.

```
set append [true | false]
```

The source file will be copied to the destination file name.

```
set copy
```

The source file will be deleted after the file has been moved or copied.

```
set delete
```

Set the mailbox to which the results will be returned. The mailbox should be in standard internet format, for example: "deschon@isi.edu".

```
set mailbox <mailbox string>
```

Set the FTP transfer mode.

```
set mode [stream | block | compress]
```

The source file will be deleted after it has been copied.

```
set move
```

Set to true to transfer multiple files.

```
set multiple [true | false]
```

Set the port for the source/destination FTP connection.

```
set port [source | destination] <port number>
```

Set the FTP structure.

```
set structure [file | record | page]
```

Set the FTP type and format / byte size parameters. Note that a normal text file is usually "ascii", and a "binary" file is often the same as an "image" file.

```
set type [ascii|ebcdic] [nonprint|telnet|carriage-control]
or
set type [image]
or
set type [local] <byte size>
```

Set to true if the STOU command is to be used. If the STOU command is supported by the destination host, the file will be stored into a file having a unique file name.

```
set unique [true | false]
```

Set to true to display full FTP conversations for "verify" and "transfer" commands.

```
set verbose [true | false]
```

B.2.10 Source Commands

Set the source directory.

```
sdir <directory name>
```

Set the source file name.

```
sfile <file name>
```

Set the source host, user, and password.

```
shost <host name/number> <login> <password>
```

B.2.11 Status Command

Display the current parameter values.

```
status
```

B.2.12 Submit Command

Submit the current request for background FTP.

```
submit
```

BFTP prompts for the following information:

```
StartTime: <date and/or time>
ReturnMailbox: <internet mailbox>
RequestKeyword: <made-up keyword>
```

B.2.13 Time Command

Set the start time, the starting retry interval, and the maximum number of tries.

```
time <date and/or time> <minutes between tries>
     <maximum number of tries>
```

B.2.14 Transfer Command

Perform the current request in the foreground.

```
transfer
```

B.2.15 Verify Command

Make the connections now to check parameters.

```
verify
```

Appendix C: Example BFTP User Script

```
deschon.isi.edu 1% telnet hobgoblin.isi.edu 152
Trying 128.9.0.42 ...
Connected to hobgoblin.isi.edu.
Escape character is '^]'.

BFTP Server (hobgoblin.isi.edu)

Background File Transfer: For help, type '?', 'help', or 'explain'.

BFTP> prompt

Copy/Move/Delete: copy

Source --
    Host: deschon.isi.edu
    User: deschon
    Password:
    Dir: ./
    File: foo*

Destination --
    Host: venera.isi.edu
    User: deschon
    Password:
    Dir: ./temp/
    File: foo*

StartTime: Tue Oct  6 10:14:43 1987 (interval) 60 (tries) 5
ReturnMailbox: deschon@isi.edu
RequestPassword:

BFTP> set multiple true
BFTP> status
    Request type: COPY

    Source --
        Host: 'deschon.isi.edu'
        User: 'deschon'
        Pass: SET
        Acct: ''
        Dir: './'
        File: 'foo*'
        Port: 21

    Destination --
        Host: 'venera.isi.edu'
```

```
            User: 'deschon'
            Pass: SET
            Acct: ''
            Dir: './temp/'
            File:'foo*'
            Port: 21

        Structure: file, Mode: stream, Type: ascii, Format: nonprint
        Multiple matching: TRUE
        Return mailbox: 'deschon@isi.edu', Password: SET
        Remaining tries: 5, Retry interval: 60 minutes

        Start after Tue Oct  6 10:14:43 1987.

   BFTP> submit
   Checking parameters...

   Request bftp560538880 submitted to run at 10:14 Oct 6.

   BFTP> quit
   bye
   Connection closed by foreign host.
   deschon.isi.edu 2%
```

Appendix D: Sample BFTP Notification Message

```
   Received-Date: Tue, 6 Oct 87 10:15:52 PDT
   Date: Tue, 6 Oct 87 10:15:47 PDT
   From: root (Operator)
   Posted-Date: Tue, 6 Oct 87 10:15:47 PDT
   To: deschon
   Subject: BFTP Results: bftp560538880

   Request bftp560538880 submitted to run at 10:14 Oct 6.

     Tue Oct  6 10:15:22 1987: starting...

       Request type: COPY
       Source: deschon.isi.edu-deschon-XXX--21-./-foo*
       Destination: venera.isi.edu-deschon-XXX--21-./temp/-
       Stru: F, Mode: S, Type: A N, Creation: STOR
       Multiple matching: TRUE
       Return mailbox: 'deschon@isi.edu', Password: SET
       Remaining tries: 5, Retry interval: 60 minutes

   Connect to: deschon.isi.edu, 21
   deschon.isi.edu ==> 220 deschon.isi.edu FTP server (Version 4.7
                           Sun Sep 14 12:44:57 PDT 1986) ready.
   Connect to: venera.isi.edu, 21
   venera.isi.edu ==> 220 venera.isi.edu FTP server (Version 4.107
                           Thu Mar 19 20:54:37 PST 1987) ready.
   deschon.isi.edu <== USER deschon
   deschon.isi.edu ==> 331 Password required for deschon.
   deschon.isi.edu <== PASS XXX
   deschon.isi.edu ==> 230 User deschon logged in.
   venera.isi.edu <== USER deschon
   venera.isi.edu ==> 331 Password required for deschon.
   venera.isi.edu <== PASS XXX
   venera.isi.edu ==> 230 User deschon logged in.
   deschon.isi.edu <== CWD ./
   deschon.isi.edu ==> 200 CWD command okay.
   venera.isi.edu <== CWD ./temp/
   venera.isi.edu ==> 250 CWD command successful.
   deschon.isi.edu <== PORT 128,9,1,56,4,106
   deschon.isi.edu ==> 200 PORT command okay.
   deschon.isi.edu <== NLST foo*
   deschon.isi.edu ==> 150 Opening data connection for /bin/ls
                           (128.9.1.56,1130) (0 bytes).
   deschon.isi.edu ==> 226 Transfer complete.
   deschon.isi.edu <== PASV
   deschon.isi.edu ==> 502 PASV command not implemented.
   venera.isi.edu <== PASV
```

```
venera.isi.edu ==> 227 Entering Passive Mode (128,9,0,32,6,200)
deschon.isi.edu <== PORT 128,9,0,32,6,200
deschon.isi.edu ==> 200 PORT command okay.
deschon.isi.edu <== RETR foo
venera.isi.edu <== STOR foo
deschon.isi.edu ==> 150 Opening data connection for foo
                        (128.9.0.32,1736) (0 bytes).
deschon.isi.edu ==> 226 Transfer complete.
venera.isi.edu ==> 150 Openning data connection for foo
                        (128.9.1.56,20).
venera.isi.edu ==> 226 Transfer complete.
venera.isi.edu <== PASV
venera.isi.edu ==> 227 Entering Passive Mode (128,9,0,32,6,201)
deschon.isi.edu <== PORT 128,9,0,32,6,201
deschon.isi.edu ==> 200 PORT command okay.
deschon.isi.edu <== RETR foo1
venera.isi.edu <== STOR foo1
deschon.isi.edu ==> 150 Opening data connection for foo1
                        (128.9.0.32,1737) (4 bytes).
deschon.isi.edu ==> 226 Transfer complete.
venera.isi.edu ==> 150 Openning data connection for foo1
                        (128.9.1.56,20).
venera.isi.edu ==> 226 Transfer complete.
deschon.isi.edu <== QUIT
venera.isi.edu <== QUIT

  Tue Oct  6 10:15:39 1987: completed successfully.
```