GFD-R-P.020 (Proposed Recommendation) W. Allcock, Editor
Argonne National Laboratory
April 2003
Revised April 2003

GridFTP: Protocol Extensions to FTP for the Grid

Status of this Memo
This document is in full conformance with all provisions of GFD-C.1.

Conventions used in this document
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [5].

Abstract
This document fully describes the GridFTP protocol. This protocol builds on RFC 959 "FILE TRANSFER PROTOCOL (FTP)", RFC 2228 "FTP Security Extensions", RFC 2389 "Feature negotiation mechanism for the File Transfer Protocol", the IETF draft draft-ietf-ftpext-mlst-16 "FTP Extensions", which are incorporated by reference. Additionally, the following extensions are defined:

SPAS	Striped Passive	This command is analogous to the PASV command, but allows an array of host/port connections to be returned. This enables STRIPING, that is, multiple network endpoints (multi-homed hosts, or multiple hosts) to participate in the transfer.
SPOR	Striped Port	This command is analogous to the PORT command, but allows an array of host/port connections to be sent. This enables STRIPING, that is, multiple network endpoints (multi-homed hosts, or multiple hosts) to participate in the transfer.
ERET	Extended Retrieve	This is analogous to the RETR command, but it allows the data to be manipulated (typically reduced in size) before being transmitted.
ESTO	Extended Store	This is analogous to the STOR command, but it allows the data to be manipulated (typically reduced in size) before being stored.
SBUF	Set TCP Buffer Size	Allows the TCP buffer size to be set explicitly

allcock@mcs.anl.gov

ABUF	Auto-negotiate TCP Buffer Size	Allows the selection of an algorithm to use to automatically determine the appropriate TCP buffer size.
DCAU	Data Channel Authentication	RFC 2228 establishes a way to use the gss on the control channel, but not on the data channel. We have added an extension to allow authentication on the data channel as well to prevent data from being hijacked.

Options have been added as shown below:

RETR: Options have been added that allow the specification of the number of TCP streams to be used between a pair of network endpoints (PARALLELISM) and the layout of data when written to multiple network endpoints (STRIPING).

Appropriate feature responses (FEAT) have been defined so that a client may determine if an implementation supports these commands. Commands allowing a selection of algorithms, such as ERET, ESTO, and ABUF, should also implement responses to the HELP command listing available algorithms.

A new mode has been defined:

EBLOCK (Extended block): is key to enabling many of the features in this specification. It sends data in blocks, where a block consists of 8 bits of flags, a 64-bit length, a 64-bit offset, and then "length" bytes of data. This enables out of order reception, which is needed for PARALLEL and STRIPED data transfers.

This combination of features will allow secure, fast, efficient, flexible, and extensible data transfer and data access.

Table of Contents

1. Authorship

This specification is the result of an effort on the part of many people. They are listed below in alphabetical order. If there is an omission, it is unintentional:

W. Allcock	Argonne National Laboratory	Editor, initial architecture discussions and first draft of protocol.
J. Bester	Argonne National Laboratory	Initial architecture discussion and first draft of protocol, one of the primary developers on the Globus implementation
J. Bresnahan	Argonne National Laboratory	Initial architecture discussion and first draft of protocol, one of the primary developers on the Globus implementation
S. Meder	Argonne National Laboratory	Initial architecture discussion and first draft of protocol., one of the primary developers on the Globus implementation
P. Plaszczak	Argonne National Laboratory	Reviewer of subsequent drafts; co-author of Globus Java client implementation
S. Tuecke	Argonne National Laboratory	Chief Architect and first draft of protocol.

Beyond the people specifically listed above, there were many people who helped via discussions of various points and informal reviews of some sections of the document. These things cannot get done without input from many, many people and to all of you everywhere, who contributed even a single comment, we thank you.

2. Introduction

2.1. Background

In Grid environments, access to distributed data is typically as important as access to distributed computational resources. Distributed scientific and engineering applications require:

* *transfers* of large amounts of data (terabytes or petabytes) between storage systems, and
* *access* to large amounts of data (gigabytes or terabytes) by many geographically distributed applications and users for analysis, visualization, etc.

The standard data transfer and access protocols most typically used in the Internet – namely FTP and HTTP – are lacking key features necessary to Grid applications. Further, multiple storage archive systems (HPSS, DPSS, SRB, etc.) have implemented specialized interfaces to provide additional features, but only the API's are available, not the underlying protocols, and thus interoperability between such systems is problematic. This has led to a fragmented Grid storage community. Users who wish to access different storage systems are forced to use multiple protocols and/or APIs, and it is difficult to efficiently transfer data between these different storage systems.

We propose a common data transfer and access protocol called GridFTP that provides secure, efficient data movement in Grid environments. This protocol, which extends the standard FTP protocol, provides a superset of the features offered by the various Grid storage systems currently in use. We chose the FTP protocol because it is the most commonly used protocol for bulk data transfer on the Internet, and of the existing candidates from which to start (http, DPSS, HPSS, SRB, etc.) ftp comes closest to meeting the needs of Grid applications. The GridFTP protocol includes the following features:

* Grid Security Infrastructure (GSI) and Kerberos support
* Third-party control of data transfer
* Parallel data transfer (multiple TCP steams between 2 network endpoints)
* Striped data transfer (1 or more TCP streams between m network endpoints on the sending side and n network endpoints on the receiving side (including cases where m and n may be different)
* Partial file transfer
* Manual/Automatic control of TCP buffer/window sizes
* Support for reliable and restartable data transfer
* Integrated instrumentation

2.2. Motivation

There are already a number of storage systems in use by the Grid community. These storage systems have been created in response to specific needs for storing and accessing large datasets. They each focus on a distinct set of requirements and provide distinct services to their clients.

For example, some storage systems (DPSS, HPSS) focus on high-performance access to data and utilize parallel data transfer streams and/or striping across multiple servers to improve performance. Other systems (DFS) focus on supporting high-volume usage and utilize dataset replication and local caching to divide and balance server load. The SRB system connects heterogeneous data collections and provides a uniform client interface to these repositories, and also provides metadata for use in identifying and locating data within the storage system. Still other systems (HDF5) focus on the structure of the data, and provide client support for accessing structured data from a variety of underlying storage systems.

Unfortunately, most of these storage systems utilize incompatible and often unpublished protocols for accessing data, and therefore require the use of their own client libraries to access data. This effectively partitions the datasets available to Grid applications. Applications that require access to data stored in different storage systems must either choose to only use a subset of storage systems, or must use multiple methods to retrieve data from the various storage systems. FTP is generally available as an option, but is not a viable choice for many Grid applications without extensions.

One approach to breaking down partitions created by these mutually incompatible storage system protocols is to build a layered client or gateway which can present the user with one interface, but which translates requests into the various storage system protocols and/or client library calls. This approach is attractive to existing storage system providers because it does not require them adopt support for a new protocol. But it also has significant disadvantages, including:

* Performance: Costly translations are often required between the layered client and storage system specific client libraries and protocols. In addition, it can be challenging to efficiently transfer a dataset from one storage system to another.

* Complexity: Building and maintaining a client or gateway that supports numerous storage systems is considerable work. In addition, staying up to date as each storage system independently evolves is very difficult. This is further

exacerbated by the need to provide support for multiple client languages, such as C/C++, Java, Perl, Python, shells, etc.

It would be mutually advantageous to both storage providers and users to have a common level of interoperability between all of these disparate systems: a common, but extensible, underlying data transfer protocol. Storage providers would gain a broader user base, because their data would be available to any client. Storage users would gain access to a broader range of storage systems and data. In addition, these benefits can be gained without the performance and complexity problems of the layered client or gateway approach.

2.3. **Terminology**

* Parallel transfer: A data transfer between two network endpoints that uses multiple TCP streams.
* Striped transfer: A data transfer between m networks endpoints on the sending side and n network endpoints on the receiving side. This could be multi-homed hosts, or multiple hosts (a cluster).
* Data Node: In a striped data transfer, a data node is one of the network endpoints returned in the SPAS command, or one of the network endpoints sent in the SPOR command.
* DTP: The Data Transfer Process establishes and manages the data connection. The DTP can be passive or active.
* PI: The Protocol Interpreter. The user and server sides of the protocol have distinct roles implemented in a user-PI and a server-PI.
* Features: A response from a server indicating it supports a set of specified functionality. This is in accordance with RFC 2389.
* Options: A command to a server defining alternative behavior. This is in accordance with RFC 2389.

3. **The Extensions**

3.1. Summary

This section describes the extensions to RFC 959. These extensions consist of commands, options, features, and a new mode.

3.2. **Commands**

3.2.1. Striped Passive (SPAS)

This extension is used to establish a vector of data socket listeners for each stripe of the data. To simplify interaction with the parallel data transfer extensions, the SPAS MUST only be done on a

control connection when the data is to be stored onto the file space served by that control connection. The SPAS command requests the FTP server to "listen" on a data port (which is not the default data port) and to wait for one or more data connections, rather than initiating a connection upon receipt of a transfer command. The response to this command includes a list of host and port addresses the server is listening on. This command MUST always be used in conjunction with the extended block mode.

Syntax
The syntax of the SPAS command is:

```
spas = "SPAS" <CRLF>
```

Responses
The server-PI will respond to the SPAS command with a 229 reply giving the list of host-port strings for the remote server-DTP or user-DTP to connect to.

```
spas-response = "229-Entering Striped Passive Mode" CRLF
                1*(<SP> host-port CRLF)
                229 End
```

The format of the host-port is as follows:
h1,h2,h3,h4,p1,p2

Where this represents the concatenation of a 32 bit IP address and a 16 bit TCP port address. h1 through h4 represents the 4 fields in a dotted IPV4 IP address transmitted as decimal numbers in character string representation. h1 is the high order 8 bits of the IP address. p1 is the high order 8 bits of the TCP port. To form the IP address, it would be h1.h2.h3.h4. To determine the TCP port it would be p1*256 + p2.

Where the command is correctly parsed, but the server-DTP cannot process the SPAS request, it must return the same error responses as the PASV command.

OPTS for SPAS
There are no options in this SPAS specification, and hence there is no OPTS command defined.

3.2.2. Striped Data Port (SPOR)

This extension is to be used as a complement to the SPAS command to implement striped third-party transfers. To simplify interaction with the parallel data transfer extensions, the SPOR MUST only be done on a control connection when the data is to be retrieved from the file

space served by that control connection for a third-party transfer. This command MUST always be used in conjunction with the extended block mode.

Syntax
The syntax of the SPOR command is:

SPOR 1*(<SP> <host-port>) <CRLF>

The host-port sequence in the command structure MUST match the host-port replies to a SPAS command.

Responses
The server-PI will respond to the SPOR command with the same response set as the PORT command described in the ftp specification.

OPTS for SPOR
There are no options in this SPOR specification, and hence there is no OPTS command defined.

Implementation Note:
The FTP protocol defines multi-line responses, but not multi-line commands. We have attempted to maintain this model. As a result, the SPOR command could potentially represent a VERY long string for the command. Implementations MUST be aware of this and prepared to deal with it.

3.2.3. Extended Retrieve (ERET)

The extended retrieve extension is used to request that a retrieve be done with additional processing on the server. This command is an extensible way of providing server-side data reduction or other modifications to the RETR command. This command is used in place of OPTS to the RETR command to allow server side processing to be done with a single round trip (one command sent to the server instead of two) for latency-critical applications.

Syntax
The syntax of the ERET command is

ERET <SP> <module-name>="<module-params>" <SP> <resource-name><CRLF>

module-name ::= <unique string identifying the module>
module=params ::= <module specific opaque string>

The module parameters are enclosed in double quotes. If the params contain double quotes, they MUST be escaped with a back slash (\").

If a back slash is contained in the params it MUST be escaped with a backslash (\\).

The resource name may be omitted, if the module does not require it.

The command should be parsed and a module selected based on matching the module name token. This module is passed the module parameters verbatim, which it must parse and then act on.

All implementations of this specification SHOULD implement the following Partial File Transfer ERET module:

```
ERET <SP> PFT="<offset>,<length>" <filename>

offset::= string representation of a positive 64 bit integer
length::= string representation of a positive 64 bit integer
```

Note that the offset specified here is the offset in the file and is not related to the offset specified in the MODE E header, which is the offset in the transfer over the wire.

Responses

The response to the ERET command should be per RFC 959 for the RETR command. Additionally, if the module specified is not recognized, a 501 MUST be returned and the text SHOULD include a list of available modules. If the module selected cannot parse the parameters a 502 MUST be returned and the text SHOULD identify proper syntax.

Options

There are no options in this ERET specification, and hence there is no OPTS command defined.

Alternative Syntax for ERET

In a previous version of this specification, an alternative syntax for the ERET command had been proposed. This syntax has been widely implemented. A server implementing this protocol MAY choose to honor the former syntax to assist in migration. However, any new modules MUST be implemented using the syntax listed above.

The alternative format of the ERET command is

```
ERET <SP> <retrieve-mode> <SP> <filename>

retrieve-mode ::= P <SP> <offset> <SP> <size>
offset ::= 64 bit integer
size ::= 64 bit integer
```

Extended Retrieve Modes

Partial Retrieve Mode (P): A section of the file will be retrieved from the data server. The section is defined by the starting offset and extent size parameters.

3.2.4. Extended Store (ESTO)

The extended store extension is used to request that a store be done with additional processing on the server.

Syntax
The syntax of the ESTO command is

```
ESTO <SP> <module-name>="<module-params>" <SP> <resource-
name><CRLF>

module-name ::= <unique string identifying the module>
module=params ::= <module specific opaque string>
```

The module parameters are enclosed in double quotes. If the params contain double quotes, they MUST be escaped with a back slash (\"). If a back slash is contained in the params it MUST be escaped with a backslash (\\).

The resource name MAY be omitted, if the module does not require it.

The command should be parsed and a module selected based on matching the module name token. This module is passed the module parameters verbatim, which it must parse and then act on.

All implementations of this specification SHOULD implement the following Partial File Transfer ESTO module:

```
ESTO <SP> PFT="<offset>,<length>" <filename>

offset::= string representation of a positive 64 bit integer
length::= string representation of a positive 64 bit integer
```

Note that the offset specified here is the offset in the file and is not related to the offset specified in the MODE E header, which is the offset in the transfer over the wire.

Responses
The response to the ESTO command should be per RFC 959 for the STOR command. Additionally, if the module-name specified is not recognized, a 501 MUST be returned and the text SHOULD include a list of available modules. If the module selected cannot parse the parameters a 502 MUST be returned and the text SHOULD identify proper syntax.

Options
There are no options in this ESTO specification, and hence there is no OPTS command defined.

Alternative Syntax for ESTO
In a previous version of this specification, an alternative syntax for the ESTO command had been proposed. This syntax has been widely implemented. A server implementing this protocol MAY choose to honor the former syntax to assist in migration. However, any new modules MUST be implemented using the syntax listed above.

The alternative format of the ESTO command is

```
ESTO <SP> <store-mode> <filename> <CRLF>
store-mode ::= A <SP> <offset>
offset ::= 64 bit Integer
```

Store Modes
Adjusted store (A): The data in the file is stored with offset added to the file pointer before storing the blocks of the file. In extended block mode, this value is added to the offset in the extended block header, and may be a positive or negative value. In block, compressed, or stream modes modes, the offset is added to the implicit offset of 0 for the beginning of the data.

3.2.5. Set Buffer Size (SBUF)

This extension adds the capability of a client to set the TCP buffer size for subsequent data connections to a value. This replaces the server-specific commands SITE RBUFSIZE, SITE RETRBUFSIZE, SITE RBUFSZ, SITE SBUFSIZE, SITE SBUFSZ, and SITE BUFSIZE

Syntax
The syntax of the SBUF command is:

```
sbuf = SBUF <SP> <buffer-size>

buffer-size ::= <number>
```

The buffer-size value is the TCP buffer size in bytes. The TCP window size should be set accordingly by the server.

Response Codes
If the server-PI is able to set the buffer size state to the requested buffer-size, then it will return a 200. Note: Even if the SBUF is accepted by the server, an error may occur later when the data connections are actually created.

3.2.6. AutoNegotiate Buffer Size (ABUF)

This extension allows the invocation of an algorithm to determine and set the TCP buffer size. No specific algorithms are defined here, but support is provided in the protocol for the arbitrary addition of algorithms. Any algorithm added should have an associated FEAT response defined listing it as an available module, and an associated HELP response for each module describing parameter syntax.

Syntax
The syntax of the ABUF command is

```
ABUF <SP> <module-name>="<module-params>"<CRLF>

module-name ::= <unique string identifying the module>
module=params ::= <module specific opaque string>
```

The module parameters are enclosed in double quotes. If the params contain double quotes, they MUST be escaped with a back slash (\"). If a back slash is contained in the params it MUST be escaped with a backslash (\\).

The command should be parsed and a module selected based on matching the module name token. This module is passed the module parameters verbatim, which it must parse and then act on.

Response Codes
If the server-PI is able to set the buffer size state to the calculated buffer-size, then it will return a 200 and SHOULD include the buffer size set in the text. If the algorithm is not identified, a 501 MUST be returned and the text SHOULD include a list of available algorithms. If the algorithm selected can not parse the parameters a 502 MUST be returned and the text SHOULD identify proper syntax. Note: Even if the server accepts the ABUF, an error may occur later when the data connections are actually created.

3.2.7. Data Channel Authentication (DCAU)

This extension provides a method for specifying the type of authentication to be performed on FTP data channels. This extension may only be used when the control connection was authenticated using RFC 2228 Security extensions.

Syntax
The format of the DCAU command is

```
DCAU <SP> <authentication-mode> <CRLF>
```

```
authentication-mode ::= <no-authentication>
                      | <authenticate-with-self>
                      | <authenticate-with-subject>

no-authentication ::= N
authenticate-with-self ::= A
authenticate-with-subject ::= S <subject-name>

subject-name ::= string
```

Authentication Modes

* No authentication (**N**)
  No authentication handshake will be done upon data connection establishment.
* Self authentication (**A**)
  A security-protocol specific authentication will be used on the data channel. The identity of the remote data connection will be the same as the identity of the user which authenticated to the control connection.
* Subject-name authentication (**S**)
  A security-protocol specific authentication will be used on the data channel. The identity of the remote data connection MUST match the supplied **subject-name** string.

The default data channel authentication mode is A for FTP sessions which are RFC 2228 authenticated. If the security handshake fails, the server must return the error response 432 (Data channel authentication failed).

3.3. **Features**

RFC 2389 provides for the addition of the FEAT and OPTS commands to allow for the negotiation of feature sets. The following new feature names are to be included in the FTP server's response to FEAT if it implements the following sets of functionality

* **PARALLEL**: The server supports MODE E, and can accept and initiate multiple TCP connections for a transfer
* **MODE-E-RESTART**: The server supports MODE E, the restart markers, and restart semantics as described in this document.
* **MODE-E-PERF**: The server supports MODE E and the performance markers as described in this document.
* **STRIPING**: The server supports the SPOR and SPAS commands, the RETR options, and extended block mode as described in this document.
* **ESTO**: The server implements the ESTO command as described in this document.

* **ERET:** The server implements the ERET command as described in this document.
* **SBUF:** The server implements the SBUF command as described in this document.
* **ABUF:** The server implements the ABUF command as described in this document.
* **DCAU:** The server implements the DCAU command as described in this document, including the requirement that data channels are authenticated by default, if RFC 2228 authentication is used to establish the control channel.

Features that allow module selection, such as ESTO, ERET, and ABUF SHOULD implement HELP responses that list the available modules.

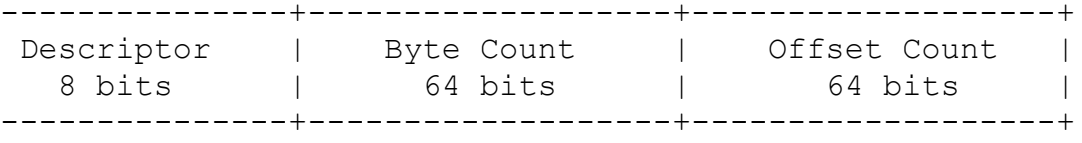
3.4. Extended Block Mode

The striped and parallel data transfer methods described above requires an extended transfer mode to support out-of-sequence data delivery, and partial data transmission per data connection. The extended block mode described here extends the block mode header to provide support for these as well as large blocks, and end-of-data synchronization. Clients indicate that they want to use extended block mode by sending the command:

```
MODE <SP> E <CRLF>
```

on the control channel before a transfer command is sent. The structure of the extended block header is:

3.4.1. Extended Block Header



The descriptor codes are indicated by bit flags in the descriptor byte. Six codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte. See section 2.4.2 for further information on the use of these bits.

Code	Meaning
128	This block is End Of Record (EOR) (Legacy)
64	This block contains the End of Data Count (EODC), which is the number of End of Data (EOD) markers (see bit 8 below) that MUST be received
32	Suspected errors in data block
16	Data block is a restart marker (Legacy)

8 This block is the End Of Data (EOD) marker for this link
4 Sender will close the data connection

Note that while this is modeled after the BLOCK mode descriptor, BLOCK mode and Extended BLOCK mode are completely independent modes (data channel protocols) and have no relationship to each other. With this encoding, more than one descriptor-coded condition may exist for a particular block. For instance, 64 (EODC), 8 (EOD), and 4 (CLOSE) could all be set simultaneously. However, there are also nonsensical encodings, for instance code 16 does not make sense with 128, since a restart marker can't be the end of a record. We leave deciding what are acceptable combinations to the implementer. The implementation must generate an error if a flag is set that it does not know how to interpret. Some additional protocol is added to the extended block mode data channels, to properly handle end-of-file detection in the presence of an unknown number of data streams.

* When no more data is to be sent on a given data channel, then the sender will mark the last block, or send a zero-length block after the last block with the EOD bit (8) set in the extended block header on that data channel.
* After receiving an EOD the data connection can be cached for use in a subsequent transfer. To signify that the data connection will be closed the sender sets the close bit (4) in the header on the last message sent.
* The sender communicates end of file by sending an EOF message to all servers receiving data. The EOF message format is described below.

An example will help to illustrate how this works. A sender intends to send multiple files to the same receiver and wishes to avoid the overhead of re-establishing the connections. The receiver listens on its port, and the sender opens 3 data connections, A, B, and C. For the first transfer, all three data channels are used, an EOD is sent on each, and one EODC of 3 is sent. No CLOSE is sent, so all three channels may be kept open. Either end could also choose to arbitrarily close them, but for our example they are cached (held open). On the second transfer, only channels A and B are used. An EODC of 2 is sent (on either channel, but only one may be sent). An EOD is sent on A and B. A CLOSE must be sent on channel B and the connection closed. The reason for this is to avoid a race condition. Suppose that we just sent the EOD on A and B, and the EODC of 2, but no close and B is heavily congested and the EOD gets dropped several times. Now you start a new transfer that uses A and C. You send very little data and send EOD on C, which arrives before B. The CLOSE prevents this race condition.

3.4.2. Extended Block EODC Header (code 64 set in descriptor)

EOF Descriptor. The EOF header descriptor has the same definition as the regular data message header described above.

EOD Count Expected. This 64 bit field represents the total number of data connections that will be established with the server receiving the file. This number is used by the receiver to determine it has received all of the data. When the number of EOD messages received equals the number represented by the "EOD Count Expected" field the receiver has hit end of file.

Simply waiting for EOD on all open data connections is not sufficient. It is possible that the receiver reads an EOD message on all of its open data connects while an additional data connection is in flight. If the receiver were to assume it reached end of file it would fail to receive the data on the in flight connection. Note that in a multi-node transfer each receiving node MUST receive EXACTLY one EODC count reply on an arbitrarily selected data channel. How the EODC count is consolidated is left as an implementation detail.

3.4.3. EOF Handling in Extended Block Mode

If you are in either striped or parallel mode, you will get exactly one EOF on each SPAS-specified ports (stripes). Hosts in extended block mode must be prepared to accept an arbitrary number of connections on each SPOR port before the EOF block is sent.

3.5. Options

3.5.1. Options to RETR

The options described in this section provide a means to convey striping and transfer parallelism information to the server-DTP. For the RETR command, the Client-FTP may specify a parallelism and striping mode it wishes the server-DTP to use. These options are only used by the server-DTP if the retrieve operation is done in extended block mode. These options are implemented as RFC 2389 extensions.

The format of the RETR OPTS is specified by:

```
retr-opts     = "OPTS" <SP> "RETR" [<SP> option-list] CRLF
option-list   = [ layout-opts ";" ] [ parallel-opts ";" ]
```

```
layout-opts   = "StripeLayout=Partitioned"
              | "StripeLayout=Blocked;BlockSize=" <block-size>
parallel-opts = "Parallelism=" <starting-parallelism> ","
                               <minimum-parallelism>  ","
                               <maximum-parallelism>

block-size           ::= <number>
starting-parallelism ::= <number>
minimum-parallelism  ::= <number>
maximum-parallelism  ::= <number>
```

3.5.1.1. Layout Options

The layout option is used by the source data node to send sections of the data file to the appropriate destination stripe. The various StripeLayout parameters are to be implemented as follows:

Partitioned
A partitioned data layout is one where the data is distributed evenly on the destination data nodes. Only one contiguous section of data is stored on each data node. A data node is defined here as a single host-port mentioned in the SPOR command

Blocked
A blocked data layout is one where the data is distributed in round-robin fashion over the destination data nodes. The data distribution is ordered by the order of the host-port specifications in the SPOR command. The block-size defines the size of blocks to be distributed.

3.5.1.2. Parallelism Options

The parallelism option is used by the source data node to control how many parallel data connections may be established to each destination data node. This extension option provides for both a fixed level of parallelism, and for adapting the parallelism to the host/network connection, within a range. If the starting-parallelism option is set, then the server-DTP will make starting-parallelism connections to each destination data node. If the minimum-parallelism option is set, then the server may reduce the number of parallel connections per destination data node to this value. If the maximum-parallelism option is set, then the server may increase the number of parallel connections to per destination data node to at most this value.

Responses
The responses to the OPT command are defined in RFC 2389. A 200 reply is returned for normal successful commands. A 501 is returned for permanent failures, a 451 for transient failures or failures

based on configuration. If the command is not recognized a 500 or 502 will result. The server COULD choose to respond with a 501 if the requested number of sockets was too large.

4. Declarative Specifications

4.1. Minimum Implementation

The extensions described in this document are designed to provide a data access and transport mechanism that is secure, fast, reliable, flexible, and extensible. However, not all applications require all these features and it is desirable that they still be able to be "part of the grid". This means, that in fact, none of the extensions described here are required for the minimum implementation. Our recommendation for a minimum implementation is as recommended in RFC 959 with the addition of the RFC 2228 Security extensions. Clear text passwords simply are no longer acceptable. We have listed the details below:

Per RFC 959:

TYPE:	ASCII Non-print
MODE:	Stream
STRUCTURE:	File, Record
COMMANDS:	USER, QUIT, PORT, TYPE, MODE, STRU,
COMMANDS:	RETR, STOR, NOOP (these commands with default values only)

The default values for transfer parameters are:

TYPE:	ASCII Non-print
MODE:	Stream
STRU:	File

All hosts must accept the above as the standard defaults.

Per RFC 2228:
COMMANDS:AUTH , ADAT, MIC, CONF, ENC

4.2. Recommended Implementation

In order to gain all the benefits and to fully take advantage of the grid, we recommend the following for a full featured implementation. Note that there are some commands, modes, features, etc, that are being deprecated as they are seldom implemented and in some cases simply no longer apply:

RFC 959, FILE TRANSFER PROTOCOL (FTP), J. Postel, R. Reynolds (October 1985)
Commands used by GridFTP

USER	PASS	ACCT	CWD	CDUP	QUIT
REIN	PORT	PASV	TYPE	MODE	RETR
STOR	STOU	APPE	ALLO	REST	RNFR
RNTO	ABOR	DELE	RMD	MKD	PWD
LIST	NLST	SITE	SYST	STAT	HELP
NOOP					

Features used by GridFTP
   Type: ASCII, Image
   Mode: Stream, EBlock
   Structure: File structure

RFC 2228, FTP Security Extensions, Horowitz, M. and S. Lunt (October 1997)

Commands used by GridFTP
   AUTH
   ADAT
   MIC
   CONF
   ENC
Features used by GridFTP
   GSSAPI authentication

RFC 2389, Feature negotiation mechanism for the File Transfer Protocol, P. Hethmon , R. Elz (August 1998)

Commands used by GridFTP
   FEAT
   OPTS

FTP Extensions, R. Elz, P. Hethmon (September 2000)

Commands used by GridFTP
   SIZE
Features used by GridFTP
   Restart of a stream mode transfer

5. Security Considerations

Security is one of the key considerations for the grid and what makes FTP as defined by RFC 959 unacceptable for use today. GridFTP was designed with security in mind from the start and was, in fact, the driving force that started this effort. While we will retain anonymous FTP, and support for the USER and PASS commands, we strongly discourage their use, particularly PASS. We incorporate the GSS API extensions defined in RFC 2228, with both GSI and Kerberos bindings.

6. Known Issues

6.1. Unidirectional data transfer in EBLOCK mode:

Currently if you are in MODE E (EBLOCK) mode, PASV must be paired with STOR, and PORT must be paired with RETR. In other words, the direction of the connection on the data channels must go from the sending (RETR) to the receiving (STOR) side. While this works, it raises the following issues:

1) The current FTP protocol does not have this restriction.
2) Firewalls: It can help you traverse some firewalls more easily to be able to set the direction to connect out from behind the firewall.
3) A mixture of partial gets/puts currently requires two control channel connections. This is less than ideal.

The restriction is necessary because in the reverse situation end of file cannot be reliably determined, allowing for the possibility of lost data. As discussed above, the sender has to send an EOD on each connection and an EOD count. If the receiver were making connections, it would be possible for n connections to have been formed, all data sent on those n connections, the EOD's sent on each connection and the EODC with a count of n sent, while the receiver has another connection in flight. No EOD would be received on this connection and thus EOF would not be properly determined.

At first blush, this problem seems quite simple to solve. However, when arbitrary number and timing of streams, multi-node transfers, cached data connections, data layout on the receiving side, and the fact that the listener does not know what host or how many are making connections to its port are taken into account, it is quite difficult.

6.2. Order dependency between PASV/SPAS and STOR/RETR:

In simple terms, the sequence of events in a GET or PUT operation as defined by RFC 959, is as follows:

1. One end of the transfer listens on a port.
2. The other end forms a connection to that port
3. The other end is told to STOR (write) a file. Again, the filename is an argument to the STOR command.
4. One end is told to RETR (read) a file, where the filename is an argument to RETR

This sequence of events imposes an unnecessary, and in some instances, significant limitation. This limitation is that the connection must be established before the file to be transferred has

been identified. This prevents any decision about the connection being made based on the filename. The ability to make decisions based on the filename allow will large installations to do load balancing and internal relocation of files transparently. If a front end server is presented with a URL, prior to the data connection being formed, then the front end could make a decision about the optimal host to service the data connection. For instance:

The host specified in the URL may be down for maintenance and its files are being hosted elsewhere. The front end could simply could direct the new host to form the data connection.

A heavily accessed file may be present on multiple backend servers and the front end can choose which one to service this request based on current load.

Several solutions to this problem have been proposed:

A new response should be defined for the PASV command: This response would a “delayed IP/Port”. This response would indicate that the IP/Port information would be returned as the first intermediate response in the STOR/RETR command. Then when the STOR/RETR command is received, a decision can be made about which host should form the data connection based on the filename/URL provided. With this information now available, connections can be established and normal STOR/RETR behavior can follow. This functionality would be turned on and off via some mechanism, perhaps an OPT or SITE command.

Redefine the state machine to allow PORT/PASV and STOR/RETR in any pairs, but unordered: Currently, the state machine is such that the STOR/RETR command knows that the data connection MUST already exist and therefore it can immediately begin transmission. If instead the state machine were redefined so that a state of “OK TO BEGIN TRANSMISSION” were defined and that state was reached by receiving one each of PORT/PASV and STOR/RETR, then there would no longer be an ordering restriction.

Introduce a new command called the Pre Transfer (PRET) command (see further description below in the section on new commands under consideration). This command would allow arbitrary state information to be set for a single transfer before any other command were issued. This option has the advantage that state information other than the filename could be provided. File size is one option that comes to mind. However, this system allows information not anticipated today to be made available. The disadvantage is that this requires additional state and complicates the RFC 959 state machine. In the absence of a PRET command, the standard RFC 959 state machine is used.

6.3. Pipelining of commands & reuse of eblock data channels:

In order to get maximal efficiency when issuing multiple RETR/ERET/STOR/ESTO commands, in addition to reusing the data channels, you would also want to pipeline the issuing of commands. That is, for example, while the data for one RETR is still being sent by the server, the client could issue another RETR command. This would, in theory, allow the server to keep the data channel pipes full. As soon as it finishes sending the data for the first RETR, it can immediately start sending the data for the next RETR, at the same time as it sends the control channel response to the first RETR.

The issue here is making sure that the receiving end can figure out where the data for the first RETR ends, and the second begins. The obvious complication is when there are multiple data channels.

6.4. Support for disk resource management:

Is there a need for protocol support that addresses disk resource management needs such as verifying and reserving available disk space, advanced reservations, requesting that a file be maintained on disk for a minimum specified period of time (pinning), etc? The current belief is that this is not necessary. A higher level service will interact with disk resource management services (if any) and get space allocated, files pinned or staged, etc, and then direct the transfer service (GridFTP) to move the files once this is all in place.

7. Appendix I: Restarting

In general, opaque restart markers passed via the block header should not be used in extended block mode. This capability is provided for back compatibility with BLOCK mode. Instead, the destination server should send extended data marker responses over the control connection, in the following form:

```
extended-mark-response = "111" <SP> "Range Marker" <SP>
                         <byte-ranges-list>

byte-ranges-list       = <byte-range> [ *("," <byte-range>) ]
byte-range             = <start-offset> "-" <end-offset>

start-offset         ::= <number>
end-offset           ::= <number>
```

The byte ranges in the marker are an incremental set of byte ranges which have been stored to disk by the data server. The complete

restart marker is a concatenation of all byte ranges received by the client in 111 responses.

The client MAY combine adjacent ranges received over several range responses into any number of ranges when sending the REST command to the server to restart a transfer.

For example, the client, on receiving the responses:

```
        111 Range Marker 0-29
        111 Range Marker 30-89
```

may send, equivalently,

```
        REST 0-29,30-89
        REST 0-89
        REST 30-59,0-29,60-89
```

to restart the transfer after those 90 bytes have been received. The server MAY indicate that a given range of data has been received in multiple subsequent range markers. The client MUST be able to handle this. For example:

```
        111 Range Marker 30-59
        111 Range Marker 0-89
```

is equivalent to

```
        111 Range Marker 30-59
        111 Range Marker 0-29,60-89
```

Similarly, the client, if it is doing no processing of the restart markers, MAY send redundant information in a restart.

8. Appendix II: Performance Monitoring

In order to monitor the performance of extended block mode transfer, an additional preliminary reply MAY be transmitted over the control channel. This reply is of the form:

```
      extended-perf-response  =
      "112-Perf Marker" CRLF
            <SP> "Timestamp:" <SP> <timestamp> CRLF
            <SP> "Stripe Index:" <SP> <stripe-index> CRLF
            <SP> "Stripe Bytes Transferred:" <SP> <byte count> CRLF
            <SP> "Total Stripe Count:" <SP> <stripe count> CRLF
      "112 End" CRLF
```

```
timestamp              = <number> [ "." <digit> ]
stripe-index           = <number>
byte count             = <number>
stripe count           = <number>
```

All perf-line facts represent an instantaneous state of the transfer at the given timestamp. The meanings of the facts are:

Timestamp - The time at which the server computed the performance information. This is in seconds since the epoch ((00:00:00 UTC, January 1, 1970).

Stripe Index - the index (range of 0 to n where n is the number of stripes on the STOR side of the transfer) that this marker pertains to.

Stripe Bytes Transferred - The number of bytes which have been received on this stripe.

Total Stripe Count - The total number of stripes (network endpoint pairs) participating in this transfer.

A transfer start time can be specified by a perf marker with 'Stripe Bytes Transferred' set to zero. Only the first marker per stripe can be used to specify the start time of that stripe. Any subsequent markers with 'Stripe Bytes Transferred' set to zero simply indicates no data transfer over the interval.

A server should send a 'start' marker for each stripe. A server should also send a final perf marker for each stripe. This is a marker with 'Stripe Bytes Transferred' set to the total transfer size for that stripe.

9. Appendix III: RFCs Considered During Development

The RFCs listed below were considered for inclusion of their functionality in the GridFTP protocol. There are other RFCs that are related in some way to the FTP protocol, but were either purely informational in nature, or were a form of File Transfer protocol, but completely different from the protocol defined in RFC 959. We chose the set of features that best fit the needs of Grid applications, and then added our own extensions as necessary.

RFC0959 File Transfer Protocol Oct-85
Comments: The primary basis of the GridFTP protocol. We chose the common features (STOR, RETR, PORT, PASV), and features that were defined, but not often implemented (third party transfer).

RFC2773 Encryption using KEA and SKIPJACK Feb-00
Comments: We chose to use existing encryption within Globus at the SSL level.

RFC2640 Internationalization of FTP Jul-99
Comments: Defines how to use UniCode characters in FTP. This is a good idea and is under consideration for future addition, but right now there is no definite requirement.

RFC2428 FTP Extensions for IPv6 and NATs Sep-98
Comments: Defines new commands (EPRT and EPSV) to allow for arbitrary addressing schemes. IPV4 and IPV6 are defined, but addition of other schemes is straight forward. This is a good idea and is under consideration for future addition, but right now there is no definite requirement. If implemented, it is likely that we would map the other commands such as PORT, PASV, SPAS, and SPOR to EPRT and EPSV transparently.

RFC2389 Feature negotiation mechanism for FTP Aug-98
Comments: The FEAT and OPTS extensions as defined in the RFC are incorporated in the GridFTP protocol.

RFC2228 FTP Security Extensions Oct-97
Comments: The AUTH, ADATA, PROT, PBSZ, CCC, MIC, CONF, ENC, and 6yz replies as defined in this RFC are incorporated in the GridFTP protocol.

RFC1639 FTP Operation Over Big Address Records (FOOBAR) Jun-94

RFC1545 FTP Operation Over Big Address Records (FOOBAR) Nov-93
Comments: It was felt that the method described in RFC 2428 was a better way of dealing with non-IPV4 addressing schemes.

RFC1068 Background File Transfer Program (BFTP) Aug-88
Comments: We consider a reliable file transfer service based on the GridFTP protocol to be a key service for the Grid. Material in this RFC is one source of input for design of such a service.

10. GGF Copyright Statement

11. GGF Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

12. References

[1] Postel, J. and Reynolds, J., " FILE TRANSFER PROTOCOL (FTP)", STD 9, RFC 959, October 1985.

[2] Hethmon, P. and Elz, R., " Feature negotiation mechanism for the File Transfer Protocol", RFC 2389, August 1998.

[3] Horowitz, M. and Lunt, S., " FTP Security Extensions", RFC 2228, October 1997.

[4] Elz, R. and Hethom, P., " FTP Extensions", IETF Draft, September 2000.