eXtended Block Mode (X-Mode) Protocol Proposal

Draft proposal, v1.2

Igor Mandrichenko, FNAL

Status of this Document

This document is a Global Grid Forum Draft. It presents a proposal for improvement of Extended Block Mode protocol to allow dynamic opening and closing of data channels during point-to-point or striped n-to-m transfers, and eliminates the uni-directional transfers feature of the protocol as described in GridFTP v1.0 draft.

This document supersedes previous version named "Modification of Extended Block Mode Proposal". It introduces new FTP data transfer mode, X mode. It incorporates the modifications to originally proposed protocol made after initial prototype implementation and recently proposed provisions for data integrity verification [crc].

Abstract

GridFTP protocol has become popular data movement tool used to build distributed grid-oriented applications. GridFTP protocol extends FTP protocol defined by RFC959 [rfc959] and other IETF documents by adding certain features designed to improve performance of data movement over wide area network, to allow the application to take advantage of "long fat" communication channels, to help build distributed data handling applications.

Several groups have developed independent implementations of GridFTP v1.0 [gftp] protocol for different types of applications. This document summarizes the experience gained by these groups and their proposals for possible protocol improvements. The goal of these improvements is to develop more robust, reliable and scalable protocol for bulk file-oriented data transfer over wide and local area networks. This document proposes one of improvements for the protocol.

Contents

Background

GridFTP protocol defines extended block data transfer mode, which can be used for parallel data transfers. The idea of transferring data over multiple concurrent TCP streams is becoming popular among computational data grid application developers. This makes extended block mode and GridFTP protocol very attractive as possible standard for data transfer mechanism.

However, extended block mode has one critical deficiency. In extended block mode, data must flow in the same direction as TCP connection establishment. In FTP standard terminology, receiving FTP server must always be passive, and sending FTP server – always active. This makes it impossible to use extended block mode with NAT, firewalls, etc.

Main reason for this limitation is difficulty in robust communication of end of file over multiple data streams. Currently, end of file is signaled using special EODC message that carries total number of open data channels. It is used to make sure sender and receiver saw the same number of open data channels and neither one is still "in flight". Unfortunately, this method works only when data sender is active. Also, it does not work in case of striping or many-to-many data transfers.

Another deficiency of existing extended block mode protocol is the fact that data receiver must accept all incoming data connections and is not allowed to close data channel socket or any existing data channel.

Also, for massive data transfers it becomes important to verify data integrity to prevent data loss due to transmission errors. Sending some sort of checksum value along with data makes it easier to detect transmission errors and recover from them.

The proposal is to introduce new data transfer mode – X mode. This is a modification of Extended Block mode documented in [gftp].

Proposed Solution

The proposed solution is based on the following ideas:

- Use robust handshake schema to open and close each data channel. This is achieved by introducing "READY", "CLOSE" and "BYE" messages sent in the beginning and at the end of the transfer on the *data channel* in the direction opposite to the data flow;
- Do not use EODC to send number of used data channels. Instead, send EOF message on one or more data channels open between two hosts. The same bit 64 can be used for EOF message;
- Send checksum value along with each data block so that the receiver can verify data integrity and immediately request retransmission of the block if an error is detected. The receiver sends "RESEND" message back to the sender on the same data channel but in the direction opposite to the data flow. Sender and receiver will use OPTS/FEAT mechanism to negotiate concrete type of the checksum prior to the data transfer.

Data Block Format

Data block format is almost the same as for B and E modes. The only difference is that if data integrity verification is turned on (using OPTS mechanism), each data

block is followed by checksum value calculated over the block header and data. Length of the checksum value is determined by previously negotiated checksum type. If checksum calculation is not turned on, then no checksum value is appended to the end of the block. Data block format is:

Field	**Length, bytes**	**Contents**
Descriptor	1	Block descriptor. Bits in the descriptor are: 64 – End of file (EOF) 8 – End of data (EOD) – request to close this data channel 4 – Sender will close this data channel instead of reusing it (?)
Byte count	8	Length of data
Offset	8	Offset of the block in the file
Data	<byte count>, can be 0	Data
Checksum	depends on the type, can be 0	Value of the checksum calculated over header and data

Data Channel Protocol

Proposed data channel protocol is outlined in Fig. 1 and 2 for active and passive sender cases respectively

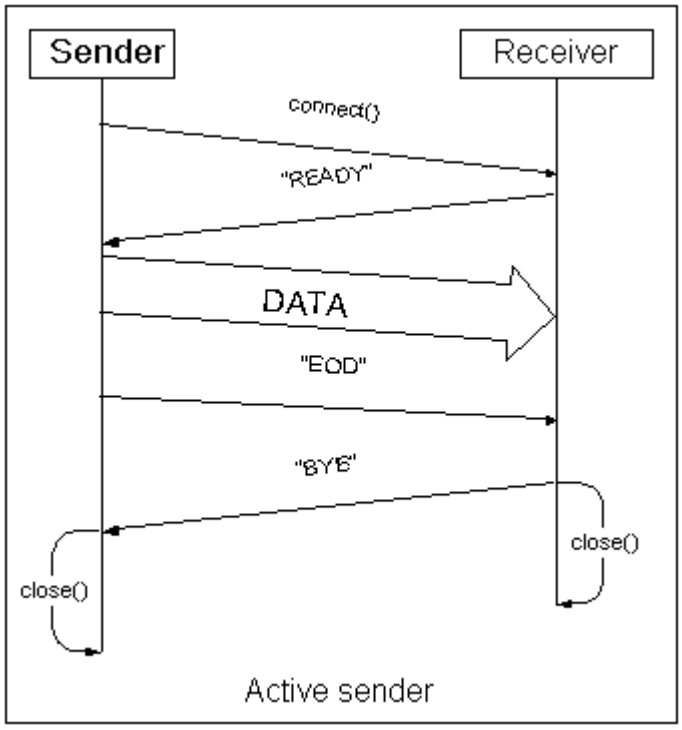

Fig. 1

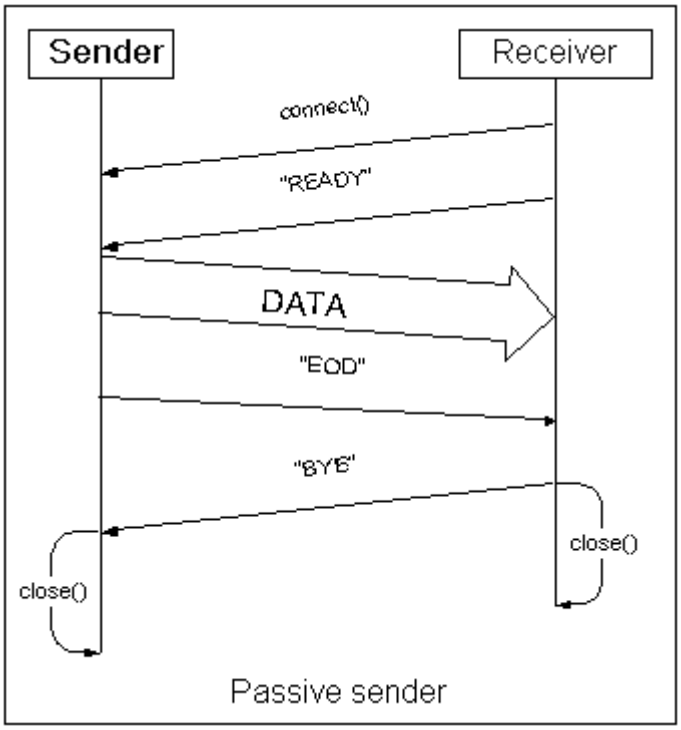

Fig. 2

Opening Data Channel

When passive data receiver accepts new incoming connection on the data socket, it must acknowledge data channel opening with "READY" message sent on newly created data channel socket to data source. Active data sender does not send any data on the data channel until it receives "READY" message. This procedure ensures that active sender and passive receiver hosts use the same number of data channels for the transaction and essentially makes it unnecessary to send channel count in EODC message.

Passive receiver may close new data socket without sending "READY" message or even stop accepting new connections. No data will be lost in such cases because the sender will not send any data before receiving "READY" message.

In case of passive sender and active receiver, there is no need for "READY" message. Sender can immediately begin sending data on newly accepted data channel socket.

In general case of many-to-many striped transfer, active peer must open at least one data channel to each passive peer host. This is necessary to make sure that , even if there is no data to be sent to or received from one of passive hosts, it does not have to wait forever for the transfer to begin.

Closing Data Channel

There are two cases when a data channel may be closed under normal circumstances:

- There is no more data to send on the channel, i.e. the sender has reached end of file

- Either sender or receiver closes one or more data channels in the middle of transfer, e.g. to control bandwidth utilization

Data Channel Closed by the Sender

Before closing a data channel socket (either at the end of the file or in the middle of the transfer), data sender (active or passive) must send EOD message as defined by extended block mode protocol on the data channel. Data receiver acknowledges EOD message with "BYE" message sent back on the data channel. Data sender may choose to wait for "BYE" message to make sure the receiver successfully received all data sent over the data channel. Failure to send "BYE" message to the sender should not be considered an error by the receiver as the sender may choose not to wait for data channel closure confirmations. After sending "BYE" message the receiver may close the data channel or keep it open to reuse in future transfers. Likewise, after receiving "BYE" the sender may choose to close the data channel or keep it open.

Data Channel Closed by the Receiver

If the receiver wishes to close a data channel in the middle of the transfer, it must send "CLOSE" message on the data channel (see Fig. 3). After sending "CLOSE" message, the receiver must continue receiving the data on the data channel until it receives EOD block. After receiving EOD block, the receiver sends "BYE" message on the channel.

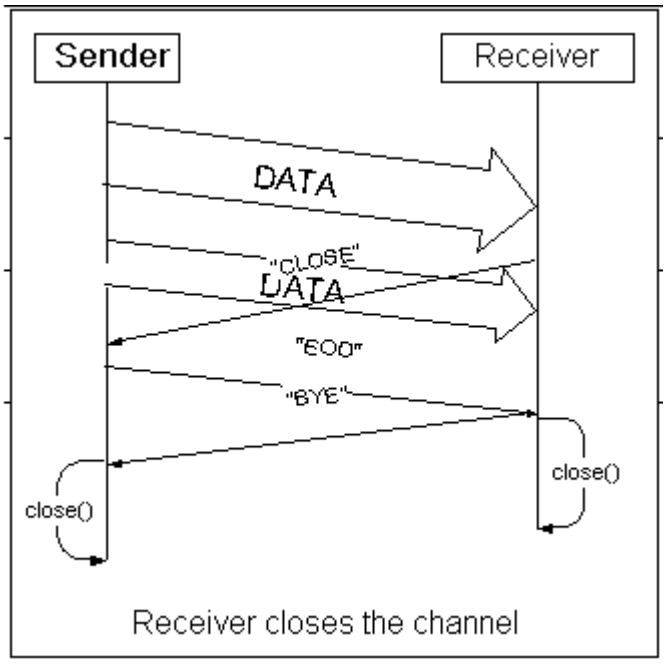

Fig. 3

Data Retransmission

If the receiver detects an error in a block transmission, it can request that the sender resends the block. To request block retransmission the receiver sends "RESEND" command on the same data channel where the erroneous block was received:

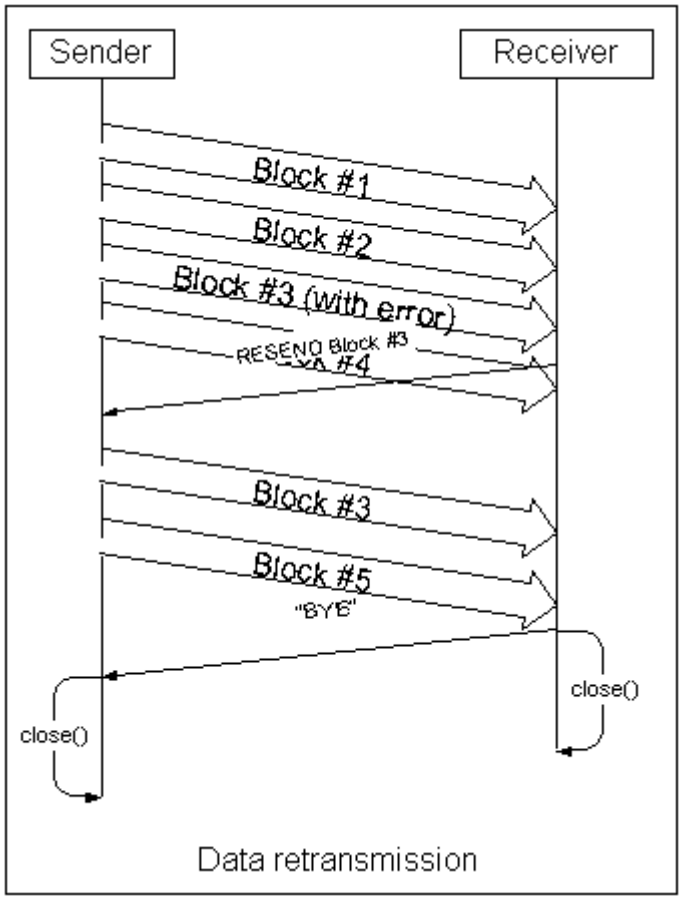

Fig. 4

After detecting a transmission error in one of data blocks and sending "RESEND" command, the receiver should continue receiving data.

It is possible that the bad block will be retransmitted *after* EOD is received. The receiver should not send final "BYE" and close the data channel until it receives all blocks it requested to be retransmitted.

Host Pairs

In most general case of striped transfers, data is sent from N *sender hosts* to M *receiver hosts.* Therefore, there are N*M sender-receiver *host* pairs. Each host pair may open zero or more data channels (see Fig. 4).

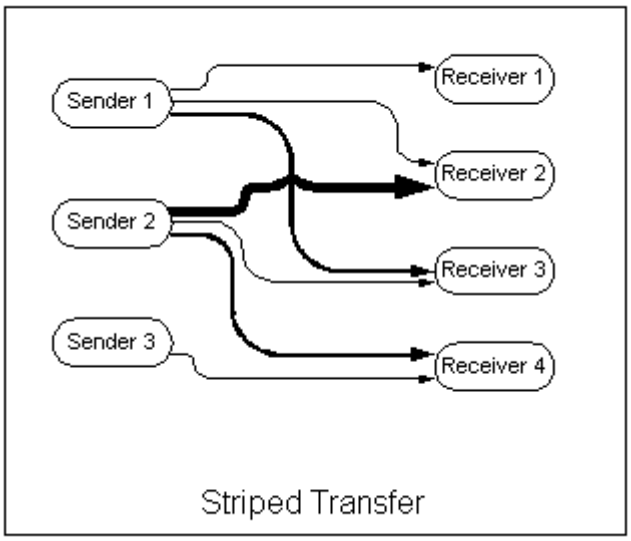

Fig. 4

The protocol allows for dynamic management of such resources as network bandwidth and socket file descriptors by allowing hosts in each pair to open and close data channels dynamically during data transfer without any data loss.

End of File Communication

End of file is signaled by sending (possibly empty) block with EOD and EOF flags set in the block descriptor. The transfer between individual sender and receiver hosts is considered finished successfully after last data channel between them is closed with EOD and the receiver host received at least one EOF block on at least one data channel established between the two hosts. In general case of many-to-many striping, EOF block must be sent on at least one data channel for every sender-receiver pair. EOF communication is described in more details for each type of host.

Active Receiver

After receiving EOF block from a sender host, active data receiver host must not try to open any new data channels to that sender host. It must continue receiving data on all previously open data channels until it receives EOD block on the channel. The receiver host may try to open new data channels to other sender hosts, those it has not received EOF from. In case of striped transfer, the receiver must attempt to open at least one data channel to each sender host. As long as at least one data channel to at least one of sender hosts was open successfully, failure to initiate other channels should not be considered an error by the receiver.

The transfer is considered finished successfully by active receiver after all data channels are closed and at least one EOF block was received from each sender host.

Passive Receiver

Passive receiver host must be receiving data on all open channels until it receives EOD on all channels with EOF on at least one of them. When EOF is received on one of data channels, passive receiver is allowed to stop accepting new data channel connections.

The transfer is considered finished successfully after all open data channels were closed with EOD and at least one EOF block was received by each receiving host.

Passive Sender

Passive sender sends EOD on all open data channels with EOF bit set on at least one data channel per receiver host. In case when it is impossible for the sender to distinguish between connections coming from different receiving hosts, sender may simply send EOF on all open data channels.

The transfer is considered successfully finished when all data was sent and all data channels were closed and the receiver acknowledged all channel closures with "BYE" messages.

Active Sender

Active sender sends EOD on all open data channels and EOF at least on one per receiving host. Sender must not send EOF on any data channel until it receives "READY" on all open channels.

In case of striped transfer, the sender must open at least one data channel to each receiver host and send at least EOD and EOF block to each host even if there is no data to be sent to the host.

The transfer is considered successfully finished when all data was sent and all data channels were closed and the receiver acknowledged all channel closures with "BYE" messages.

Dynamic Resource Allocation

For some applications, it is desired that such resources as network bandwidth, CPU power and open I/O channels (file descriptors) can be dynamically allocated and reallocated between concurrent transfers. Proposed protocol allows for new data channels to be open and closed in the middle of transfer without data loss or corruption. There are provisions for active or passive sender or receiver to open, close or refuse to open new data channel at any time during transfer.

Active Sender

Active sender can control number of open data channels by opening and closing them at any time. The receiver acknowledges new data channel with "READY" message that allows the sender to start using the new channel. At any time active sender can close any data channel after sending EOD block and optionally receiving "BYE" as the acknowledgement.

Active Receiver

Active receiver can control number of open data channels by opening and closing them at any time. The sender may or may not send any data on newly open channel. Once the channel is open by the active receiver, it may close it at any time, but only after sending "CLOSE" message and receiving EOD block. The receiver must keep the channel open and continue receiving data until EOD block is received on the channel.

Passive Sender

Passive sender, naturally, cannot open new data channels, so it cannot increase bandwidth utilization by adding new channels. It can only decrease bandwidth utilization by:

- Closing data socket port thus refusing new data connections
- Closing newly opened data connection before sending any data on the channel
- Sending EOD and closing the data channel
- Sending EOD, waiting for "BYE" and closing the data channel

Existing data channel can be closed at any time after sending EOD block and optionally waiting for "BYE".

Passive Receiver

Passive receiver can decrease bandwidth utilization by:

- Closing data socket port and refusing new data connections
- Closing newly opened data connection before sending "READY"
- Sending "CLOSE" as a request to close the data channel

Once "READY" message was sent to the sender, passive receiver must receive all data sent on the channel until it receives EOD block or the sender closes the channel.

Data Channel Command Syntax

This section describes the format of commands sent by the data receiver on the data channel socket. General format is text terminated with carriage return, linefeed combination or just linefeed:

```
<DC command> = <keyword> [<parameters>] [<CR>] <LF>
```

Commands and their parameters are:

```
READY                    (no parameters)
```

The receiver sends READY command after the data channel is open to allow the sender to start sending the data.

```
CLOSE                    (no parameters)
```

Data receiver sends this command when it needs to close the data channel. The receiver must continue receiving data even after sending CLOSE command until it receives EOD block.

```
BYE                      (no parameters)
```

This command is sent by the receiver to allow the sender to close the data channel. It acknowledges that the receiver has successfully received all the data sent on this data channel. The sender must not close the data channel until it receives "BYE" command. The receiver closes the channel right after it sends "BYE".

`RESEND <offset> <length>`

The receiver uses this command to request retransmission of a data block. Offset and length are ASCII strings representing decimal numbers for block offset within the file and its length. The sender does not necessarily have to resend requested data in single block. It may split it into several blocks if necessary.

References

[gftp] Bill Allcock, et all, GridFTP v1.0 Draft
http://www-isd.fnal.gov/gridftp-wg/draft/GridFTPRev3.htm

[rfc959] IETF RFC959 http://www.ietf.org/rfc/rfc0959.txt?number=959

[crc] Timur Perelmutov, GridFTP Data Integrity Verification
Draft proposal
http://home.fnal.gov/~timur/gridftp/index.html