# Distributed Replicated Files Protocol Specification

Sining Ma
sma87@stanford.edu

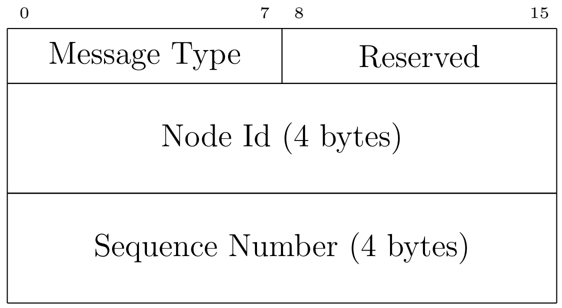
May 15, 2014

## 1 Protocol Specification

This section describes all message packets syntax, semantics. All messages bytes are in network byte order.

### 1.1 Common Header

All packets start with the same header structure.



**Message Type** The type of Message which is defined

**Reserved** This field is reserved and should always be zero.

**Node Id** This field represents the unique id for a client or server. Will talk more in Client and Server Id generation section about how this id is generated.

**Sequence Number** Each outgoing packet contains a monotonically increasing sequence number. This number wraps back to zero when this number overflows.

### 1.2 Init (Message Type 0xC0)

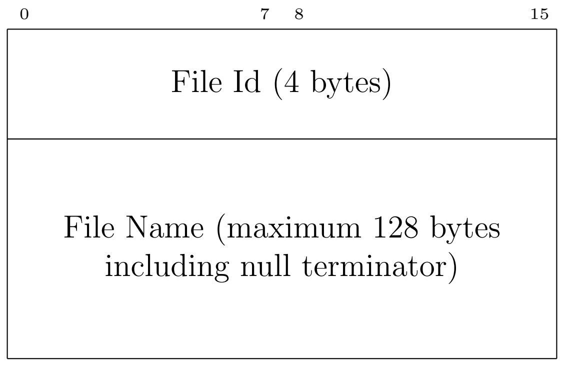
The client sends Init message when InitReplFs() is called to detect the servers.

### 1.3 InitAck (Message Type 0xC1)

Servers send InitAck messages when the server receives Init message to inform the server's existence.

## 1.4 OpenFile (Message Type 0xC2)

The client sends OpenFile message to servers when OpenFile() is called to open a new file on local client file system. Servers opens the file based on filename.

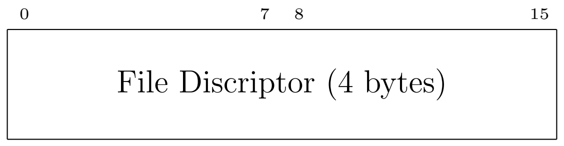


**File Id** File Id generated at client side. This Id is a monotonically increasing number.

**File Name** File name is OpenFile() argument.

## 1.5 OpenFileAck (Message Type 0xC3)

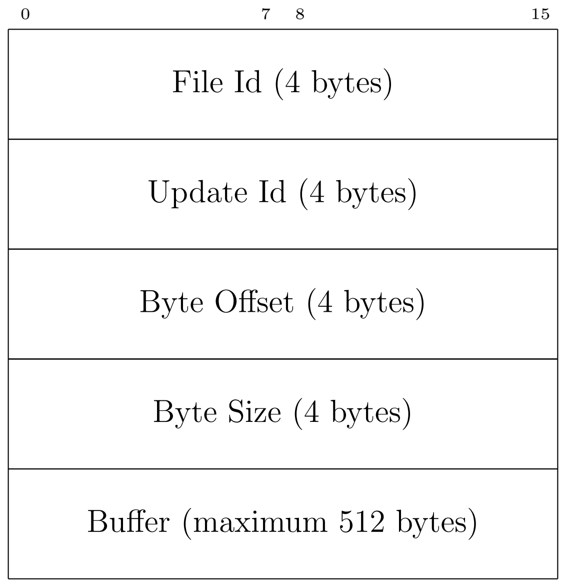
Servers send OpenFileAck messages to acknowledge OpenFile message with file id generated by the server. If the server is unavailable or error happens when open file, -1 returns.



**File Descriptor** File descriptor that is used at server side. -1 if error happens at the server side.

## 1.6 WriteBlock (Message Type 0xC4)

The client sends WriteBlock message when WriteBlock() is called to stage a contiguous chunk of data. No Ack message will be sent from servers. Before one commit, the client can do multiple WriteBlock call to update the file. Update Id is used to mark update sequence id before commit call.



**File Id** File Id for the opened file at client side.

**Update Id** In one commit, current update id is a monotonically increasing number from 0.

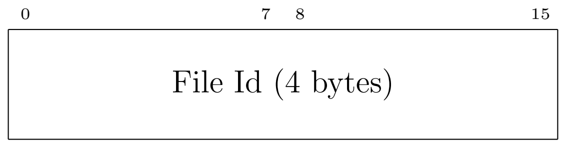
**Byte Offset** Byte offset from WriteBlock() the offset within the file to write to.

**Byte Size** Byte size from WriteBlock() the number of bytes in buffer.

**Buffer** Buffer from WriteBlock() a pointer to the data to be written.

## 1.7 Vote (Message Type 0xC5)

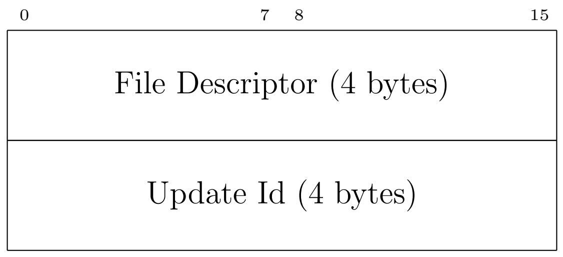
Two phase commit is used to ensure that all updates in WriteBlock are received on the server side. The client sends Vote message as the first step in commit request phase to check if all the servers receive all updates and are ready to do final commit.



**File Id** File Id for the opened file at client side.

## 1.8 VoteAck (Message Type 0xC6)

Servers send VoteAck message with an update Id which indicates the latest Update Id that the server has received successfully from the client. When the client receives this message, the client checks if update Id in the message is identical to the latest update Id that is required from the client side for this commit. If update id does not match, the client will do message retransmission. If VoteAck message returns -1 file descriptor, abort message will be sent. Will talk more about retransmission in protocol operation flow.

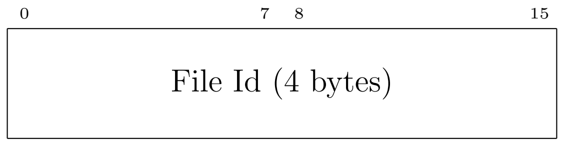


**File Descriptor** File descriptor that is used at server side. -1 if error happens at the server side.

**Update Id** The latest update that is done successfully on the servers.

## 1.9 Commit (Message Type 0xC7)

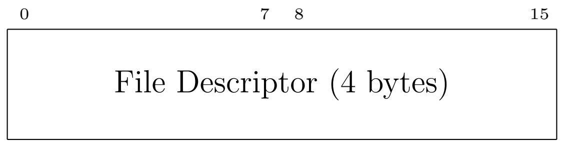
The client sends Commit message to all servers to complete and commit all updates to the file.



**File Id** File Id for the opened file at client side.

## 1.10 CommitAck (Message Type 0xC8)

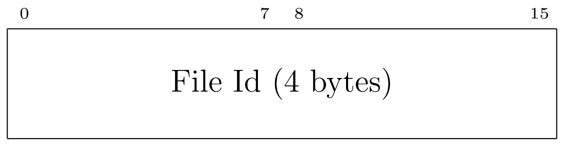
Servers send CommitAck message to the client to tell the client to complete the transaction.



**File Descriptor** File descriptor that is used at server side. -1 if error happens at the server side.

## 1.11 Abort (Message Type 0xC9)

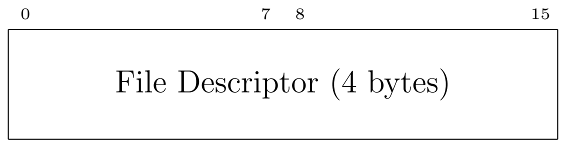
The client sends Abort message to all the servers when receive VoteAck message with -1 file descriptor. All the servers undo the transaction and rollback file update.



**File Id** File Id for the opened file at client side.

### 1.12 AbortAck (Message Type 0xCA)

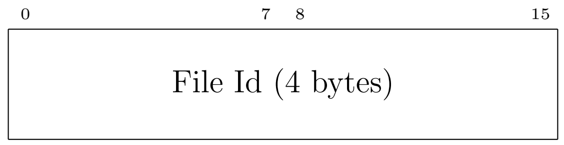
Servers sends AbortAck message to the client. This message informs the client file rollback on servers is done.



**File Descriptor** File descriptor that is used at server side. -1 if error happens at the server side.

### 1.13 Close (Message Type 0xCB)

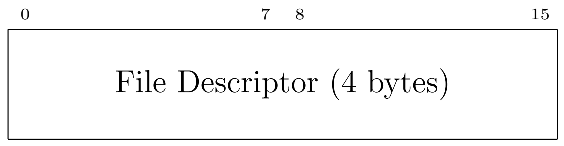
The client sends Close message to the servers to close the opened file.



**File Id** File Id for the opened file at client side.

### 1.14 CloseAck (Message Type 0xCC)

Servers send CloseAck message to indicate the file is closed.



**File Descriptor** File descriptor that is used at server side. -1 if error happens at the server side.

## 2 Protocol Details

### 2.1 Client and Server Id generation

Client and server Ids are generated randomly as a 32-bit integer with a good seed for number generator. The probability of conflict is 5.32101e-20. This is negligible. In real world settings, each client or server is assigned an unique identifier, e.g. UUID.

### 2.2 File Id generation

File Id in the client side is a monotonically increasing number starting from 0.

## 2.3 Protocol Operation Flow

- On InitReplFs(), the client sends init message to the multicast group to detect if there are enough servers count in init timeout time. If responding servers count is smaller than numServers, the client returns error. Otherwise, if there are more servers, the client selects the first numServers count, and only accepts acknowledge messages from these selected servers.

- On OpenFile(), the client sends OpenFile message to servers to open a file and waits for OpenFileAck message.

- OpenFileAck message is sent from the servers. If file Id is -1, this means that OpenFile fails and the client returns error. If response selected servers count is less than init phase servers count, the client returns error. The assumption is that there is only one client running, so no need to remember client id on servers.

- On WriteBlock() sends file update message to the servers without acknowledge. Client remembers total update count before one commit. One server updates the latest update id only when the server receives a valid WriteBlock updates. For example,The client sends update 0, 1, 2, 3 to the servers, so the client latest update id is 3. If the server receives update 0,1, but 2 is lost, the server side latest update id is 1. Then even if update 3 is received, this update is ignored and the server side latest update Id is still 1, because 2 is lost. On the client side, the client latest update id is 3. Another case is packets disorder. Packets may not be received in packets sending order. Servers cache all the updates received before receiving vote messages. Then servers start to check if next update id is received from the latest update id. If received, update latest update id number. This can solve message disorder problem.

- On Commit(), the client sends vote message to all servers, and waits for vote response. The servers receive vote message and reply VoteAck message with file descriptor and the latest update Id. If any error happens on the server side, VoteAck message return -1 file descriptor.

- The client receives many VoteAck message in vote timeout time. The first case if all servers response VoteAck message with update id match client latest update id, client sends commit message to all servers. The second case is that the client receives VoteAck message with update Id smaller than client latest update id from some servers. But the client receives all servers VoteAck messages. The client checks all servers VoteAck messages and finds the smallest update id. The client starts retransmit file updates from the smallest update Id to all servers. In this transmission phase, the servers only accept updates that is bigger than server latest update id. After retransmission, the client sends Vote message again and goes back to wait VoteAck messages (back to this step). The third case, if any server sends VoteAck with -1 file descriptor or no VoteAck in timeout time, the client sends abort message to all servers.

- The client sends Commit or Abort message to servers. This steps completes this transaction or rollback. The client waits CommitAck or AbortAck message.

- On Abort(), the client sends abort message to servers. Waits for servers AbortAck messages.
- The client receives CommitAck or AbortAck message. The client returns error if file descriptor is -1 or timeout.
- On Close(), the client sends close message to close the file. The client returns error if file descriptor is -1 or not sufficient server CloseAck message is received in timeout time.

## 2.4 Protocol Timing

### 2.4.1 InitReplFs

The client sends Init messages to servers every 200ms, and client waits for 2s. The client only accepts servers InitAck message in init phase timeout.

### 2.4.2 OpenFile

The client sends OpenFile messages to servers every 200ms. OpenFile phase timeout is 4s. The client returns -1 if not sufficient server OpenFileAck messages are received in this 4s timeout time.

### 2.4.3 Vote

The client sends Vote messages to servers every 200ms, and Vote phase timeout is 4s. If any server has no response in 4s, the client treats this case as replies VoteAck message with -1 file description. The client sends abort message.

### 2.4.4 Commit and Abort

The client sends Commit or Abort messages to servers every 200ms, this Commit or Abort timeout is 4s. If any server has no response in 4s, the client returns error but servers keep the commit that has already happened. On the server side, if any server cannot receive any commit and abort message in commit or abort phase for 4 seconds, the server rollbacks file updates automatically.

### 2.4.5 Close

The client sends Close messages to servers every 200ms, and Close phase timeout is 2s. The client returns -1 if not sufficient server CloseAck messages are received.