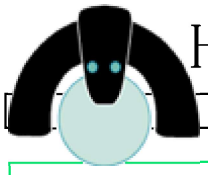

eDonkey2000 Protocol

Abstract

This document describes the eDonkey2000 (ED2K) P2P protocol and eMule Extended Protocol (EMEP).

Table of Contents

1. Introduction

ED2K protocol is used for communication between compatible clients to form a server-based peer-to-peer file-sharing network. EMEP is only used between eMule (and compatible) clients, but not by official eDonkey2000 clients.
Few words on the packet format strings used here:

```
u8      unsigned 8-bit integer
u16     unsigned 16-bit integer
u32     unsigned 32-bit integer
hash    16-bytes md4 checksum
```

```
string  16-bit integer (specifying string length) followed by the string
Taglist 32-bit integer (specifying number of tags) followed by the tags
```

Definitions

- part is a region of a file of (up to) 9500kb in length, for which there is a corresponding MD4 PartHash.
- chunk is a region of a file of (up to) 180kb in length, which can be requested from a remote client.
- block is a region of a chunk which is sent as single packet during data transfer between clients. Chunks are split into block packets for transmitting.

See also:
eDonkey2000 Tag System Overview

2. Header specification

```
+--------+--------+--------+--------+--------+--------+--------+--------+----
|protocol|            packet length           |  packet data (length bytes)
+--------+--------+--------+--------+--------+--------+--------+--------+----
```

Each eDonkey2000 packet begins with protocol opcode, followed by 32-bit unsigned integer indicating the packet length. Currently, three protocol opcodes are in use:

```
PR_ED2K  = 0xe3,  //!< Standard, historical eDonkey2000 protocol
PR_EMULE = 0xc5,  //!< eMule extended protocol
PR_ZLIB  = 0xd4   //!< Packet payload is compressed with gzip
```

Packed data always begins with 8-bit opcode, which indicates the packet content. When sending packets using PR_ZLIB protocol, the packet data (excluding the opcode) is compressed.

3. Client-Server communication

3.1 Logging into a server

In order to connect to an eDonkey2000 Server, Client sends OP_LOGINREQUEST to server.

```
OP_LOGINREQUEST = 0x01, //!< <hash>hash<u32>ip<u16>port<TagList>tags
```

The packet may contain the following tags:

```
CT_NICK         = 0x01, //!< <string>nick
CT_VERSION      = 0x11, //!< <u8>0x3c
CT_PORT         = 0x0f, //!< <u16>port
CT_MULEVERSION  = 0xfb, //!< <u32>ver
CT_FLAGS        = 0x20, //!< <u8>flags
```

Flags is a bitfield containing the following information:

```
FL_ZLIB         = 0x01, //!< zlib compression support
FL_IPINLOGIN    = 0x02, //!< Client sends its own ip during login
FL_AUXPORT      = 0x04, //!< ???
FL_NEWTAGS      = 0x08, //!< support for new-styled eMule tags
FL_UNICODE      = 0x10  //!< support for unicode
```

On a well-formed OP_LOGINREQUEST packet, the server first attempts to connect to the sending client to its reported listening port as a normal client. If the

connection attempt succeeds, and the server receives a wellformed OP_HELLOANSWER packet from the socket, the server assigns the connecting client High ID. If the connection fails for whatever reason, the client is assigned Low ID.

See also:
5 LowID Clients

The following packets are sent by the server to inform the client that the connection has been established, as well as to provide up2date information about the server:

```
OP_SERVERMESSAGE = 0x38, //!< <u16>len<len>message
OP_SERVERSTATUS  = 0x34, //!< <u32>users<u32>files
OP_IDCHANGE      = 0x40  //!< <u32>newid
```

Note:
Lugdunum 16.44+ servers send additional u32 in ID_CHANGE packet containing bitfield of supported features. Those include:

```
FL_ZLIB         = 0x01, //!< zlib compression support
FL_NEWTAGS      = 0x08, //!< support for new-styled eMule tags
FL_UNICODE      = 0x10  //!< support for unicode
```

3.2 Updating server list and information (optional)

After establishing connection with the server, the client may request additional information from the server, as well as known servers list.

```
OP_GETSERVERLIST = 0x14, //!< (no payload)
```

The server answers with the following packets:

```
OP_SERVERLIST    = 0x32, //!< <u8>count[{<u32>ip<u16>port}*count]
OP_SERVERIDENT   = 0x41, //!< <hash>hash<u32>ip<u16>port<TagList>tags
```

The latter contains the following tags:

```
CT_SERVERNAME    = 0x01, //!< <string>name
CT_SERVERDESC    = 0x0b  //!< <string>desc
```

3.2 Publishing Shared Files

After establishing connection with the server, the client must publish it's shared files using OP_OFFERFILES packet. The packet should be compressed if the server supports it to save bandwidth.

```
//! <u32>count[<count>*{<hash>filehash<u32>ip<u16>port<TagList>tags}]
OP_OFFERFILES    = 0x15,
```

The packet may contain the following tags:

```
CT_FILENAME      = 0x01,       //!< <string>name
CT_FILESIZE      = 0x02,       //!< <u32>size
CT_FILETYPE      = 0x03,       //!< <string>type
```

Note:
File type is sent in ed2k network as string. The following strings are recognized:

```
#define FT_ED2K_AUDIO    "Audio"    //!< mp3/ogg/wma      etc
#define FT_ED2K_VIDEO    "Video"    //!< avi/mpg/mpeg/wmv etc
#define FT_ED2K_IMAGE    "Image"    //!< png/jpg/gif/tiff etc
#define FT_ED2K_DOCUMENT "Doc"      //!< txt/doc/rtf      etc
#define FT_ED2K_PROGRAM  "Pro"      //!< exe/bin/cue/iso  etc
```

This packet should be sent on the following occasions:

- With full shared files list when connecting to server
- With single file whenever new shared file is added
- As empty packet, as server keep-alive packet at regular intervals
  Additionally, it is possible to indicate whether the file being shared is partial or complete using two special ip/port values:

```
FL_COMPLETE_ID   = 0xfcfcfcfc, //!< File is complete - send this as ID
FL_COMPLETE_PORT = 0xfcfc,     //!< File is complete - send this as port
FL_PARTIAL_ID    = 0xfbfbfbfb, //!< File is partial  - send this as ID
FL_PARTIAL_PORT  = 0xfbfb      //!< File is partial  - send this as port
```

Note:
Only use these ID's on newer eservers. eMule uses zlib-support to detect whether the server supports these id's too.

3.3 Performing a search

Searching in eDonkey2000 network is server-based. Client sends SEARCHR packet to server, which in turn returns one (or more) SEARCHRESULT packets. This is done over TCP, and with currently connected server only. However, optionally, the client may also send the same search request packet over UDP sockets to all known servers using GLOBSEARCH opcode, which in turn reply (over UDP) with GLOBSEARCHRES packets. <searchexpr> format specifi- cation is beyond the scope of this document. Refer to ED2KPacket::Search implementation for more information (packets.cpp)

```
OP_SEARCH        = 0x16,       //!< <searchexpr>
//! <u32>count[<count>*{<Hash>hash<u32>id<u16>port<Taglist>tags}]
OP_SEARCHRESULT  = 0x33,
```

SearchResult may contain the following tags:

```
CT_FILENAME      = 0x01,       //!< <string>name
CT_FILESIZE      = 0x02,       //!< <u32>size
CT_FILETYPE      = 0x03,       //!< <string>type
CT_SOURCES       = 0x15,       //!< <u32>numsrc
CT_COMPLSRC      = 0x30,       //!< <u32>numcomplsrc
```

4. Client <-> Client communication

4.1 get-to-know-you chit-chat

In order to initialize a connection to a remote client, first the client sends Hello packet.

```
OP_HELLO        = 0x01, //!< <hash>hash<u32>ip<u16>port<TagList>tags
```

Hello packet may contain the following tags:

```
CT_NICK         = 0x01, //!< <string>nick
CT_PORT         = 0x0f, //!< <u16>port
CT_MULEVERSION  = 0xfb, //!< <u32>ver
CT_MODSTR       = 0x55, //!< <string>modstring
CT_UDPPORTS     = 0xf9, //!< <u16>kadudpport<u16>ed2kudpport
CT_MISCFEATURES = 0xfa, //!< <u32>features bitset
```

The correct answer to OP_HELLO is OP_HELLOANSWER. The HelloAnswer packet format is identical to OP_HELLO, except that HelloAnswer also includes 8-bit hash length before hash. The value of the field must be 0x0f.
CT_MULEVERSION contains a 32-bit value with the following bits (and meanings)

```
1       2       3       4       5 [bytes]
11111111222222233333334445555555  [bits]
00000000000000001010101100000000  eMule 42g version info
    |       |       |     |    +---- Build version        (unused by eMule)
    |       |       |     +--------- Update version 6     (a = 0, g = 6)
    |       |       +--------------- Minor version 42
    |       +----------------------- Major version        (unused by eMule)
    +------------------------------- Compatible Client ID (CS_EMULE = 0x00)
```

If both of the clients are eMule-compatible, they also exchange MuleInfo packets.

```
OP_MULEINFO       = 0x01, //!< <u8>clientver<u8>protver<TagList>tags
OP_MULEINFOANSWER = 0x02  //!< <u8>clientver<u8>protver<TagList>tags
```

Note:
Both of these packets are sent using PR_EMULE.

Tags contained in MuleInfo packets:

```
CT_COMPRESSION    = 0x20, //!< u32 compression version
CT_UDPPORT        = 0x21, //!< u32 udp port
CT_UDPVER         = 0x22, //!< u32 udp protocol version
CT_SOURCEEXCH     = 0x23, //!< u32 source exchange version
CT_COMMENTS       = 0x24, //!< u32 comment version
CT_EXTREQ         = 0x25, //!< u32 extended request version
CT_COMPATCLIENT   = 0x26, //!< u32 compatible client ID
CT_FEATURES       = 0x27, //!< u32 supported features bitset
CT_MODVERSION     = 0x55, //!< <string>modversion (may also be int)
CT_MODPLUS        = 0x99, //!< mh? (Source: eMule+ Forums ... )
CT_L2HAC          = 0x3e, //!< mh? (Source: eMule+ Forums ... )
```

Feature set bitfield is 32-bit value with following bits and meanings:

```
12345678123456781234567812345678
00000100000100110011001000011110 eMule 43b
00110100000100110011001000011110 eMule 44b
11123333444455556666777788889abc
 | | |   |  |   |    |  |  |||+-- Preview
 | | |   |  |   |    |  |  ||+--- Multipacket
 | | |   |  |   |    |  |  |+---- No `view shared files' supported
 | | |   |  |   |    |  |  +----- Peercache
 | | |   |  |   |    |  +-------- Comments
 | | |   |  |   |    +----------- Extended requests
 | | |   |  |   +---------------- Source exchange
 | | |   |  +-------------------- Secure ident
 | | |   +----------------------- Data compression version
 | |  +---------------------------- UDP version
 | +------------------------------- Unicode
 +--------------------------------- AICH version (0 - not supported)
```

4.2 Reqesting a file

After completing the handshaking described in section 4.1, the client may now request a file. This is done using ReqFile packet.

```
OP_REQFILE        = 0x58, //!< <hash>hash
```

Note:
eMule ExtReqV1 adds <u16>count<count>bitarray partmap
eMule ExtReqV2 adds <u16>completesources

ReqFile is replied by OP_FILENAME and OP_FILEDESC

```
OP_FILENAME       = 0x59, //!< <hash>hash<u32>len<len>name
OP_FILEDESC       = 0x61, //!< <u8>rating<u32>len<len>comment
```

After receiving the above packets, the downloading client sends SETREQFILEID to bind the reqested file to the hash. This means the client is now bound to the requested hash, until it receives it.
FileDesc packet is sent using PR_EMULE, and does NOT contain the file hash the description belongs to, however, since it is always sent along with OP_FILENAME, in response to OP_REQFILE packet, client can thus assume the FileDesc packet belongs to the most recently requested file.

Note:
Actually, eMule extended protocol allows changing the reqested file while waiting in the queue by resending OP_SETREQFILEID at any time.

```
OP_SETREQFILEID    = 0x4f, //!< <hash>hash
```

The expected response to this packet is OP_FILESTATUS

```
OP_REQFILE_STATUS  = 0x50, //!< <hash>hash<u16>count<count>partmap
```

If at any point during the above packets the uploading client realizes it doesn't share the file requested, it may send OP_REQFILE_NOFILE. File status packet contains a part map of the available parts. This is defined as a bitfield, one bit per each ED2K part (defined by ED2K_PARTSIZE). Extra bits needed to fill up the byte are padded with zeros. The count prior to the bitset indicates the number of total parts, and thus also the number of total bits in the bitset (note: NOT the number of bytes following!). Also note that sending partmap is completely optional, and should only be sent if the file is indeed partial - if the sender has the entire file, the partmap shall be omitted.

```
OP_REQFILE_NOFILE  = 0x48, //!< <hash>hash
```

If the downloading client needs a hashset of the file, it may ask the uploader for it using OP_REQHASHSET. The expected response is OP_HASHSET.

```
OP_REQHASHSET      = 0x51, //!< <hash>hash
OP_HASHSET         = 0x52, //!< <hash>hash<u16>cnt[cnt*<hash>parthash]
```

After that, the requesting client sends OP_STARTUPLOADREQ, which must be replied either by OP_ACCEPTUPLOADREQ (if we can start uploading right away), or OP_QUEUERANKING, in case the client has been inserted into our upload queue.

```
OP_STARTUPLOADREQ  = 0x54, //!< may contain <hash>hash (emule)
OP_ACCEPTUPLOADREQ = 0x55, //!< Empty
OP_QUEUERANKING    = 0x5c, //!< <u32>queueranking
```

There is another queue ranking packet used by eMule - denoted by opcode OP_MULEQUEUERANK.

```
OP_MULEQUEUERANK   = 0x60  //!< <u16>queuerank<u16><u32><u32>empty
```

This packet contains 16-bit value as queue rank, and 10 empty bytes, such as the full packet size must be 12 bytes. This size requirement is enforced by eMule packet parser, and the sender will be disconnected if those extra bytes are not sent. Why are there those 10 extra empty bytes, and why is it enforced so strongly remains unclear. This packet is also sent using PR_EMULE protocol opcode. Note that if you'r thinking to skip over this packet and not use it, you'r so out of luck, since eMule considers clients who send OP_QUEUERANKING simply QueueFull (altough it seems to parse the packet). So if you want eMule compatibility, you need to use this half-empty packet.

4.3 Uploading file data

After receiving OP_ACCEPTUPLOADREQ, the remote client will proceed to request chunks from us. In eDonkey2000 network, a chunk size is 180k (notice the difference from partsize). The chunks are requested using OP_REQCHUNKS packet:

```
OP_REQCHUNKS      = 0x47, //!< <hash>hash[3*<u32>begin][3*<u32>end]
```

The packet contains three begin offsets and three end offsets of the requested chunks. ED2K data ranges logic defines that the end offset is exclusive, thus range (0, 0) is considered as invalidrange. If the client wishes less than three chunks, it may set the remaining ranges to (0, 0) to indicate that.

Note:

This is slightly different from HydraNode Range API, where end offset is considered inclusive.

Implementors should be warned about different behaviours of different clients when it comes to chunk requests. Namely, eMule (and compatible) clients use "rotational chunkrequest" scheme, where each REQCHUNKS packet contains one new chunk and two older chunks; for example:

```
Packet1:   0..101, 101..201, 201..301 // initial req
Packet2: 101..201, 201..301, 301..401 // sent after completing   0..101
Packet3: 201..301, 301..401, 401..501 // sent after completing 101..201
```

MLDonkeys, and some other clients, however, only send single requests:

```
Packet1:   0..101, 101..201, 201..301 // initial req
Packet2: 301..401,   0..0,     0..0   // after completing   0..101
Packet3: 401..501,   0..0,     0..0   // after completing 101..201
```

This can lead to duplicate data being sent by mldonkeys, if a rotational chunkrequest scheme is used when communicating with them. For that reason, HydraNode uses non-rotational request scheme with all clients, since all clients handle it properly.

After receiving OP_REQCHUNKS, the uploading clients starts sending data. The requested chunks are split into (up to) 10kb blocks, each transmitted as a separate packet, using OP_SENDINGCHUNK opcode. Optionally, clients supporting eMule extended protocol may use OP_PACKEDCHUNK packet (which is sent using PR_EMULE). In compressed packet, the data part of the packet is compressed using zlib (but not the packet header - this is different from OfferFiles packet, where everything after the opcode is compressed). Also, in case of a compressed packet, the end offset bytes in the packet header indicate the length of the compressed data, not the end offset of the uncompressed data. Note that compression is not required - if data compression results in larger amount of data, data should be sent uncompressed instead, using OP_SENDINGCHUNK packet.

```
OP_SENDINGCHUNK   = 0x46, //!< <hash>hash<u32>begin<u32>end<data>
OP_PACKEDCHUNK    = 0x40, //!< <hash>hash><u32>begin<u32>len<len>data
```

When the receiver decides it does not want any more data from us, it will send OP_CANCELTRANSFER packet (empty payload) to the uploader.

```
OP_CANCELTRANSFER = 0x56  //!< empty
```

In case of compressed data transfer, the entire requested chunk is first compressed, and then transmitted in 10kb chunks over time.

5 LowID Clients

LowID client is defined in ed2k network as a client who cannot receive external connection. The client can only make external connections itself. The only way to connect to a client is through a server callback, since the client is connected to

a server. It is only possible to contact LowID clients that are on same server as the connecter is. LowID client's are identifier in the network using client ID less than 0x00ffffff. This is also the reason why everywhere where there should be client IP we say it is ID. For clients with ID > 0x00ffffff, the ID indicates the IP of the remote client (in network byte order), for lower ID's, it indicates the client callback ID, as assigned by the server.

5.1 Connecting to LowID clients

Connecting to LowID client is done by sending OP_REQCALLBACK to currently connected server, including the ID of the remote client we wish to connect.

```
OP_REQCALLBACK  = 0x1c,   //!< <u32>id
```

If the client with the specified ID is connected to the server, the server sends OP_CBREQUESTED packet to the client, after which the client will attempt to contact the requester.

```
OP_CBREQUESTED  = 0x35,   //!< <u32>ip<u16>tcpport
```

If the client in question is not connected to this server, the server will respond with OP_CALLBACKFAIL.

```
OP_CALLBACKFAIL = 0x36    //!< empty
```

6 Source Exchange

Extended protocol feature added by eMule, Source Exchange allows clients to share their known sources for files.

```
OP_REQSOURCES    = 0x81,   //!< <hash>hash
OP_ANSWERSOURCES = 0x82
```

Note:
SrcExchv2 adds 16-byte user-hash to each source.
SrcExchv3 sends ID's in byteswapped format (also called "Hybrid" format in eMule code) so HighID clients with *.*.*.0 won't be falsely given LowId.

Normally, this packet is sent using PR_EMULE, but if possible, it should be compressed and sent using PR_ZLIB.

Note:
ED2K netiquette specifies that you should not request mosources from clients at intervals exceeding these:

- No more than once per 40 minutes per client in case of rare files (less than 10 sources)
- No more than once per 2 hours per client in case of common files

7 Secure Identification

Secure Identification provides means of determining the client's identity in a secure manner, and avoid the former problems regarding hash/credit stealers. In the protocol, three packets implement SecIdent:

```
OP_SECIDENTSTATE = 0x87, //!< <u8>state<u32>challenge
```

Request signature and/or publickey from remote client

```
OP_PUBLICKEY     = 0x85, //!< <u8>len<len>pubkey
```

Includes the senders public key

```
OP_SIGNATURE     = 0x86  //!< <u8>len<len>signature
```

Includes the senders signature
The signature message contents are as follows:

```
<*>pubkey<u32>challenge
```

Note:
SecIdentv2 adds u32 ip and u8 iptype The pubkey is the signature requesters public key. The message is signed with the senders own private key. The receiver can then verify the message with the senders public key.

Sample implementation of this scheme, using CryptoPP library,
CreditsDb::createCryptKey CreditsDb::loadCryptKey CreditsDb::createSignature CreditsDb::verifySignature

8 Miscellaneous TCP packets

```
OP_MESSAGE       = 0x4e,    //!< <u16>len<len>message
```

Message packets can be used to perform simple chatting between clients.

```
OP_PORTTEST      = 0xfe,    //!< Server: <u16>0x12, Client: <u16>1
```

Not actually part of ed2k protocol, OP_PORTTEST can be used (by a website, for example) to verify correct firewall configuration. Note that OP_PORTTEST may be sent both via TCP and UDP.

```
OP_CHANGEID      = 0x4d    //!< <u32>oldid<u32>newid
```

Sent when client changes it's ip address, or in case of LowID clients, changes it's ID (sent by server). Mules seem to only handle this packet, but not send it themselves.

Note:
The actual format of this packet has not yet been fully verified.

9 UDP packets

The format of UDP packets header differs slightly from TCP packet headers, namely, the size field is omitted, and opcode follows the protocol code.

9.1 eMule extended protocol: UDP reasks

As an eMule extended feature, UDP packets are used to "ping" sources every now and then to verify queue status, reask queue rankings and so on. In current implementation, eMule (and compatible) clients seem to drop clients from their queues if they haven't pinged at least once per hour. The following packets are currently in use, and are sent using protocol PR_EMULE:

```
OP_REASKFILEPING = 0x90,    //!< <hash>hash
```

By default, this packet only contains filehash the source was interested in.

Note:
UDPv3 adds 2-bytes complete-source-count.
UDPv4 adds availability partmap, as in ReqFile packet.

When UDPv4 is used, the partmap is placed before the complete source count.

```
OP_REASKACK      = 0x91,   //!< <u16>queueranking
```

Expected response to OP_REASKFILEPING, this indicates that the asking client is indeed in queue, and contains it's current queue ranking.

Note:
UDPv4 adds availability partmap, as in ReqFile packet
Similarly to OP_REASKFILEPING packet, when UDPv4 is used, the partmap is placed BEFORE the queue ranking in the packet.

```
OP_FILENOTFOUND  = 0x92,   //!< empty
```

Sent as response to OP_REASKFILEPING, indicates that the remote client is not sharing the requested file (anymore at least).

```
OP_QUEUEFULL     = 0x93,   //!< empty
```

Sent as response to OP_REASKFILEPING, indicates that the remote queue is full.

8.1 Global server queries

In Client <-> Global Server communication, UDP is used for global searches, as well as global source queries. The following packets are being used, and sent using protocol PR_ED2K:

```
OP_GLOBGETSOURCES   = 0x9a,  //!< cnt*[<hash>hash]
```

All servers support this packet, but if the ServerUDPFlags includes FL_GETSOURCES, the server allows requesting sources for multiple files at same time. When sending more than one hash, care must be taken to limit the packet size to 512 bytes, anything above that will be ignored by server.

```
OP_GLOBGETSOURCES2  = 0x94,  //!< cnt*[<hash>hash<u32>size]
```

Modified version of the above packet, this also includes file-size in the request. This can only be used with servers, which support FL_GETSOURCES2 flag.

Note:
You shouldn't query any server twice for sources during 20-minute- interval, or you might get banned.

```
OP_GLOBFOUNDSOURCES = 0x9b,  //!< <hash>hash<u8>cnt*[<u32>id<u16>port]
```

The expected response to OP_GLOBGETSOURCES or GLOBGETSOURCES2 packets, this packet contains the list of sources the server knows.

```
OP_GLOBSTATREQ      = 0x96,  //!< <u32>challenge
```

Used for "pinging" global servers, this packet requests the remote server to return info about it's current state. Servers always respond to this packet (if alive), so this can be used to determine whether the server is alive or not. As with GlobGetSources, no server should be queried more often than once every 20 minutes.
The first two bytes of the challenge (in network byte order) are always set to 0x55aa, followed by random 16bit integer.

```
OP_GLOBSTATRES      = 0x97
```

The expected response to OP_GLOBSTATREQ, this packet contains bunch of information about the server. Only the first three fields (challenge, users, files) are always present, the remainder of fields are optional (only implemented by newer server versions, and can possibly be omitted even by newer servers). The challenge is equal to the challenge sent in OP_GLOBSTATREQ packet.