The eMule Protocol Specification

Yoram Kulbak and Danny Bickson

Email: {yorkol,daniel51}@cs.huji.ac.il

Academic supervisor: Prof. Scott Kirkpatrick

DANSS (Distributed Algorithms, Networking and Secure Systems) Lab
School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel

January 20, 2005

Contents

1 Introduction

1.1 Purpose and scope

eMule is a popular file sharing application which is based on the eDonkey protocol. This report describes the network behavior of eMule and explains the basic terminology that is needed to understand the protocol. The report also gives a full specification of the eMule network protocol including an appendix which provides the message formats. The information in this document is based on an open source eMule client [2]. The purpose of the following introduction is to provide general background that will allow the reader to to read and understand this document. An extensive information source about eMule is found in [3].

1.2 Overview

The eMule network is populated with several hundreds of eMule servers and millions of eMule clients [1]. Clients should connect one server for getting network services, the server connection stays open as long as the client is in the system. The servers are performing centralized indexing services (like in Napster) and do not communicate with other servers.

Each eMule client is pre-configured with a list of servers and a list of shared files on its local file system. A client uses a single TCP connection to an eMule server for logging into the network, getting information about desired files and available clients. The eMule client also uses several hundreds of TCP connections to other clients which are used to upload and download files. Each eMule client maintains an upload queue for each of his shared files. Downloading clients join the queue at its bottom and advance gradually until they reach the top of the queue and begin downloading his file. A client may download the same file from several other eMule clients, getting different fragments from each on. A client may also upload chunks of a file which it has not yet completed downloading. Finally, eMule extends the eDonkey capabilities and allows clients to exchange information about servers, other clients and files. Note that both client and server communication is TCP based.
The server employs an internal database in which it stores information about clients and files. An eMule server doesn't store any files, it acts as a centralized index for storing information about the location of files. An additional function of the server, which is becoming deprecated, is to bridge between clients that connect through a firewall and are not able to accept incoming connections. The bridging functionality increases considerably the server load. eMule employs UDP to enhance the client's capabilities against both the server and other clients. The client's ability to send and receive UDP messages is not mandatory for the client's correct daily operation and it would function flawlessly when a firewall prevents it from sending and receiving UDP messages.

1.2.1 Client to server connection

Upon startup the client connects using TCP to a single eMule server. The server provides the client with a client ID (section 1.3) which is valid only through the client-server connection's

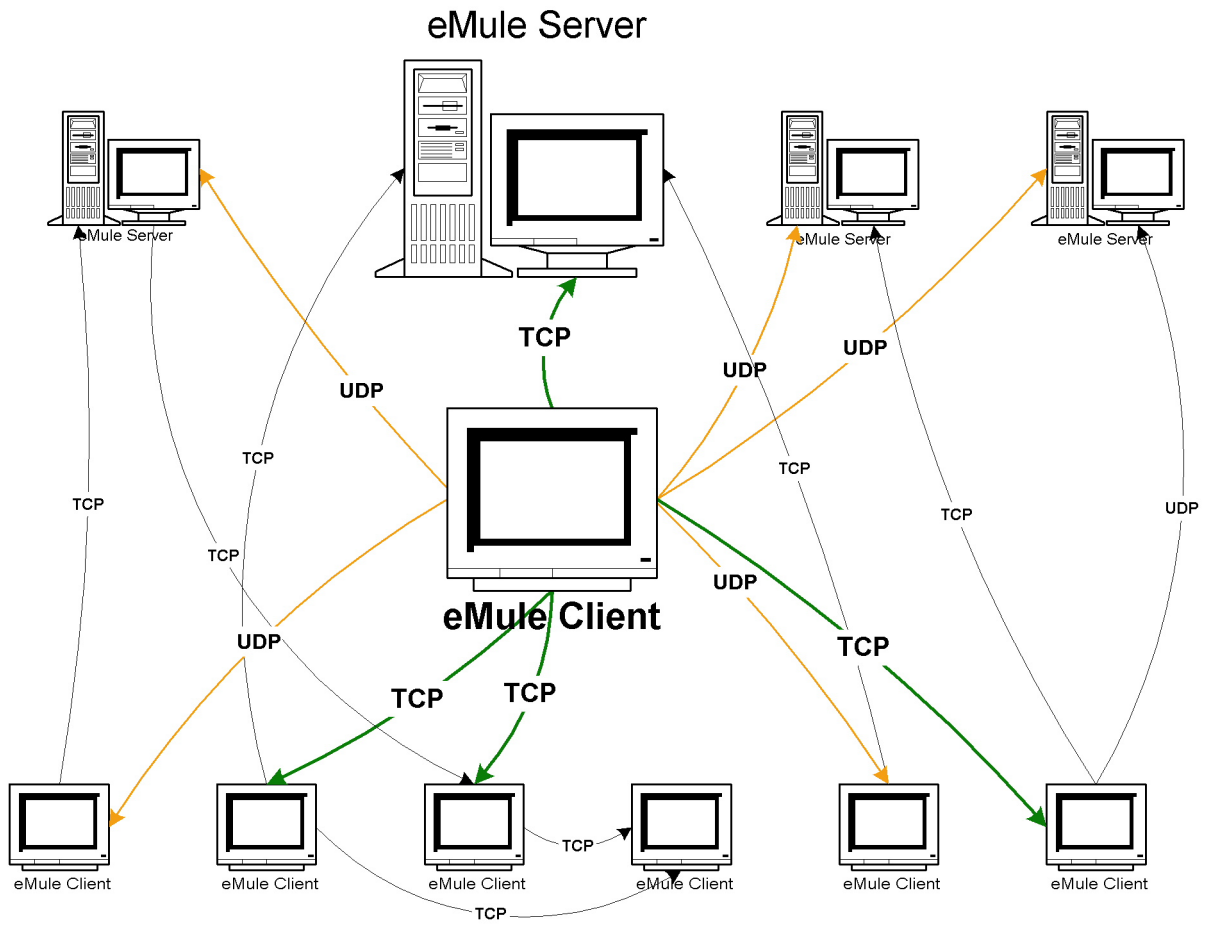

Figure 1.1: eMule high level network diagram

life time (note that when the client has high ID it will receive the same ID from all servers until its IP address changes). Following the connection establishment the client sends the server his list of shared files. The server stores the list in its internal database which usually contains several hundred thousand of available files and active clients. The eMule client also sends his download list which contains the files that it wishes to download. Section 2 provides a detailed description of the eMule client and server TCP message exchange.
After the connection is established, the eMule server sends the client a list of other clients that posses files which the connecting client wishes to download (these clients are called 'sources'). From this point on, the eMule client begins to establish connections with other clients as described in section 1.2.2 below.
Note that the client/server TCP connection is kept open during all the client's session. After the initial handshake transactions are triggered mainly by user activity: From time to time, the client sends file search requests which are replied by a search results, a search transaction is usually followed by a query for sources for a specific file, this query is replied with a list of sources (IP and port) from which the requester can download the file from.
UDP is used for communication with servers other than the server to which the client is

currently connected. The purpose of UDP messages is file search enhancement, source search enhancement and finally, keep-alive (make sure that all the eMule servers in the client's server list are valid). More details about client-server UDP message exchange can be found in chapter 3.

1.2.2 Client to client connection

An eMule client connects to another eMule client (a source) in order to download a file. A file is divided to parts which are further fragmented. A client may download the same file from several (different) clients getting different fragments from each one.
When two clients connect they exchange capability information and then negotiate the start of a download (or upload, depends on perspective). Each client has a download queue which holds a list of clients that are waiting to download files. When the eMule client's download queue is empty a download request will most probably result in a download start (unless, for example, if the requester is banned). When the download queue isn't empty a download request results in adding the requesting client to the queue. There is no attempt to serve more than a few clients in a given moment providing a minimum bandwidth of 2.4 kbytes/sec for each. A downloading client may be preempted by a waiting client with a higher queue ranking than his, in the first 15 minutes of the a download session the queue ranking of the downloading eMule client is boosted to prevent thrashing.
When a downloading client reaches the head the download queue, the uploading client initiates a connection in order to send him his needed file parts. An eMule client may be on the waiting queue of several other clients, registered to download the same file parts in each one. When the waiting client actually completes downloading the parts (from one of them) it doesn't notify all the rest that they can remove him from their queues, it will simply reject their upload attempt when it reaches the head of their queue.
eMule employs a credit system (see section 1.4) in order to encourage uploads, to prevent impersonation eMule secures the credit system using RSA public-key cryptography.
Client connections may use a set of messages not defined by the eDonkey protocol, these message are called the extended protocol. The extended protocol is used for the credit system implementation, for general information exchange (like updates of the lists of servers and sources) and to improve performance by sending and receiving compressed file fragments.
The eMule client connection uses UDP in a limited manner to periodically check the client's status on the upload queue of his peer clients while it is waiting to start downloading a file.

1.3 Client ID

The client ID is an a 4 byte identifier provided by the server at their connection handshake. A client ID is valid only through the lifetime of a client-server TCP connection although in case the client has a high ID it will be assigned the same ID by all servers until its IP address changes. Client IDs are divided to low IDs and high IDs. The eMule server will typically assigns a client with a low ID when the client can't accept incoming connections. Having a low ID restricts the client's use of the eMule network and might result in the server's rejecting the client's connection. A high ID is calculated on the basis of the client's IP address as described below. This section describes the client ID assignment and significance from the eMule protocol point of view [3]. A high ID is given to clients that allow other clients to freely connect to eMule's TCP port on their host machine (the default port number is 4662). A

client with a high ID has no restrictions in its use of the eMule network. When the server can't open a TCP connection to the client's eMule port the client is given a low ID. This happens mainly with clients that set up a firewall on their machine denying incoming connections. A client might also receive a low ID when it the following cases:

- When the client is connected through a NAT or proxy servers.
- When the server is to busy (causing the sever's reconnection timer to expire).

High IDs are calculated in the following way: assuming the host IP is X.Y.Z.W the ID will be $X + 2^8 * Y + 2^{16} * Z + 2^{24} * W$ ('big endian representation'). A low ID is always lower than 16777216 (0x1000000) I could not found any clew about how its calculated, note that when you have a low ID it differs between servers.
A low ID client has no public IP to which other clients can connect to, thus all communication must be done through the eMule server. This increases the server's computational overhead and results in reluctance of servers to accept low ID clients. Also, this means that a low ID client can't connect to another low ID client which is not on the same server because eMule doesn't support tunneling the requests between servers.
To support low ID clients a callback mechanism was introduced. Using this mechanism a high ID client can ask (through the eMule server) the low ID client to connect to it in order to exchange files.

1.4 User ID

eMule supports a credit system in order to encourage users to share files. The more files a user uploads to other clients, the more credit it receives and the faster it will advance in their waiting queues [3].
The user ID is a 128 bit (16 byte) GUID created by concatenating random numbers, the 6th and 15th bytes are not randomly generated, their values are 14 and 111 respectively. While the client ID is valid only through a client's session with a specific server the user ID (also called user hash) is unique and is used to identify a client across sessions (the User ID identifies the workstation). The user ID plays an important part in the credit system, this provides motivation for 'hackers' to impersonate to other users in order to receive the privileges granted by their credits. eMule supports an encryption scheme which is designed to prevent fraud and user impersonation. The implementation is a simple challenge-response exchange which relies on RSA public/private key encryption.

1.5 File ID

File IDs are used both to uniquely identify files in the network and for file corruption detection and recovery. Note that eMule doesn't rely on the file's name in order to uniquely identify and catalog it, a file is identified by a globally unique ID computed by hashing the file's content. There are two kinds of file IDs - the first is used mainly for generating the unique file ID, the second is useful for corruption detection and recovery [3]. .

1.5.1 File hash

Files are uniquely identified by a 128 bit GUID hash calculated by the client and based on the file's contents. The GUID is calculated by applying the MD4 algorithm [4] on the file's data. When calculating the file ID the file is divided in parts each 9.28MB long. A GUID is calculated separately for each part and then all the hashes are combined into the unique file ID. When a downloading client completes downloading a file part it calculates the part hash and compares it against the part hash sent by its peer, should the part be found corrupted, the client will try to recover from the corruption by gradually replacing bits (180kb each) of the part until the hash is calculated OK.

1.5.2 Root hash

The root hash is calculated for each part using the SHA1 algorithm, based on blocks sized 180kb each. It provides a higher level of reliability and fault recovery, mode details in the official eMule website.

1.6 eMule protocol extensions

Although eMule is completely compatible with eDonkey it implements several extensions which allow two eMule clients to provide additional functionality to their users. The extensions are focused in the client to client communication especially in the areas of security and UDP utilization. In this document all message flow diagram designate messages that are part of eMule extension in gray.

1.7 Soft and hard limits

The server configuration includes two kind of limits on the number of active users - soft and hard. The hard limit is greater equal to the soft limit. When the number of active users reaches the soft limit the server stops accepting new low ID client connections, when the user count reaches the hard limit the server is full and doesn't accept any client connection.

2 Client server TCP Communication

Each client connects to exactly one server using TCP connection. The server assigns the client an ID which will be used for to identify the client in the rest of his session with that server (A high ID client is always assigned with his IP address). The eMule GUI client requires that a server connection will be established in order to operate. The client can't be connected to several servers at the same time and nor can't it dynamically change servers without user intervention.

2.1 Connection establishment

When establishing connection to a server the client may try to connect to several servers in parallel, abandoning all but upon a successful login sequence.

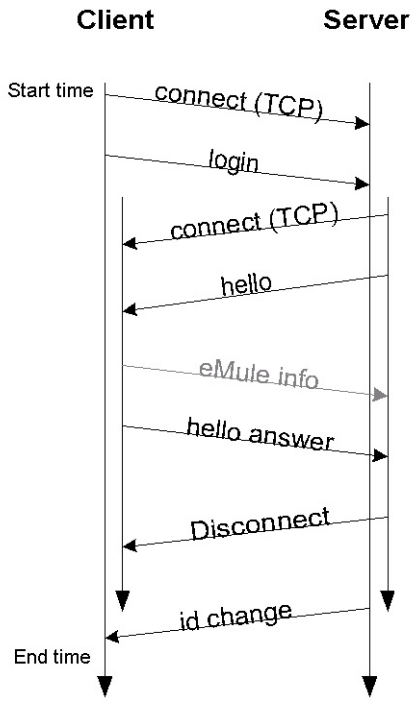

Figure 2.1: High ID login sequence

There are several possible connection establishment use cases:

1. High ID connection - the server assigns a high ID to the connecting client
2. Low ID connection - the server assigns low ID to the connecting client

3. Rejection session - the server rejects the client

There is, of course, the trivial use case in which the server is down or unreachable.

Figure 2.1 describes the message sequence that leads to a high ID connection. In this case, the client establishes a TCP connection to the server and then sends a login message to the server. The server connects using another TCP connection to the client and performs a client-to-client handshake to make sure that the connecting client has the capability to accept connections from other eMule clients. After completing the client handshake the server closes the second connection and completes the client-server handshake by sending the ID change message. You probably noticed that the eMule-info message is grayed. This is because this message is part of the eMule protocol extension (section 1.6).

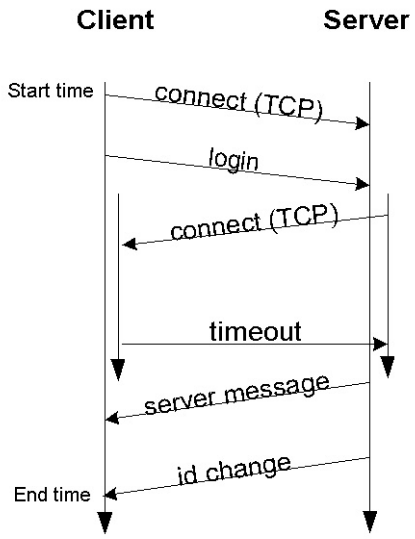

Figure 2.2: Low ID login sequence

Figure 2.2 describes the message sequence that leads to a low ID connection. In this case, the server fails to connect to the requesting client and the client is assigned with a low ID. The server message usually contains a warning like *"Warning [server details] - You have a lowid. Please review your network config and/or your settings."*
Both low and high ID handshakes complete with the ID change message which assigns the client with a client ID for its next coming session with the server.

Figure 2.3 describes the rejected session sequence. Servers might reject sessions due to the client's having a low ID or when reaching their hard capacity limit. The server message will contain a short string describing the rejection reason.

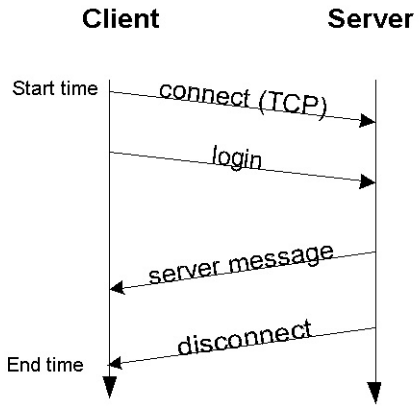

Figure 2.3: Reject session sequence

2.2 Connection startup message exchange

After a successful connection establishment the client and server exchange several setup messages. The purpose of these messages is to update both parties regarding their peer's state. The client starts by offering the server his list of shared files (see section 6.2.4), and then he asks to update his list of servers. The server sends his status and version (sections 6.2.6 and 6.2.2) and then sends his list of known eMule servers and provides some more self identification details. Finally the client asks for sources (other clients that can be accessed to download the files in his download list) and the server replies with a series of messages, one for each file in the client's download list, until all the sources list has been downloaded to the client. Figure 2.4 illustrates this sequence.

2.3 File search

The file search is initiated by the user. The operation is simple, a search request (see section 6.2.9) is sent to the server which is then answered by a search result (section 6.2.10). When there are many results, the search result message is compressed. Next, the user chooses to download one or more files, the client then requests sources for the chosen files and the server replies with a list of sources (see 6.2.12) for each of the requested files. An optional server status message may be sent by the server just before the found sources reply. The status message (section 6.2.6) contains information about the current number of users and files supported by the server. An important note is that there is a complementary sequence of UDP message which enhances the ability of the client to locate sources for his search list for more details see section 3. After verifying that sources are new, the eMule client initiates a connection attempt and adds them to its sources list. The order in which sources are contacted is the order in which they were received by the eMule client. Figure 2.5 describes the file search sequence.
The eMule client connects to sources by the order they were added to its list. There is no priority mechanism to decide to which source to connect.There is a complicated mechanism to

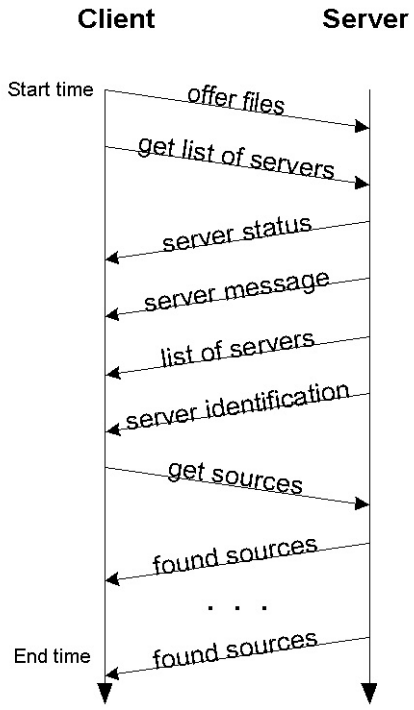

Figure 2.4: Connection startup sequence

resolve situations where the same source can be requested for downloading several files on the

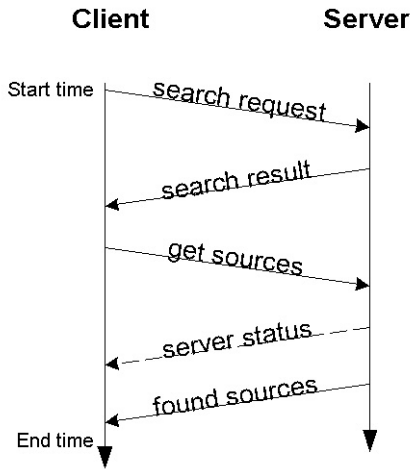

Figure 2.5: File search sequence

client's download list (Note that eMule allows only a single upload connection between clients). The selection algorithm is based on user priority specification and defaults to alphabetical ordering when no priority is specified. A details description of the handling a source which can upload more than a single file is described in the website.

2.4 Callback mechanism

The callback mechanism is designed to overcome the inability of low ID clients to accept incoming connections and thus share their files with other clients. The mechanism is simple: in case a clients A and B are connected to the same eMule Server and A requires a file that is located on B but B has a low ID, A can send the server a callback request (see section 6.2.13), requesting the server to ask B to call him back. The server, which already has an open TCP connection to B, sends B a callback requested (section 6.2.14) message, providing him with A's IP and port. B can then connect to A and send him the file without further overhead on the server. Obviously, only a high ID client can request low ID clients to call back (a low ID client is not capable of accepting incoming connections). Figure 2.6 illustrates the callback message exchange.

There is also a feature allowing two low ID clients to exchange files through their server

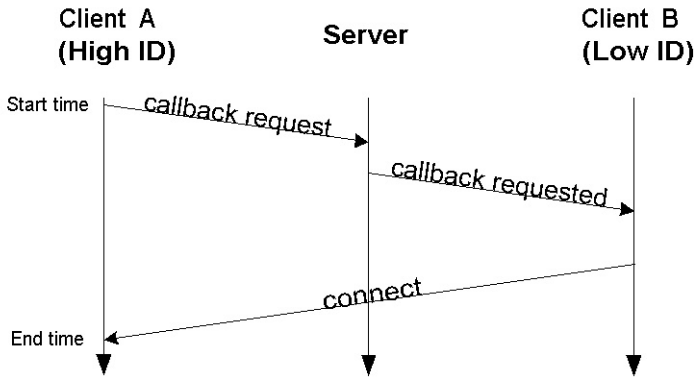

Figure 2.6: Callback sequence

connection, using the server as a relay. most of the servers no longer support this option because of the overhead it incurs on the server.

3 Client server UDP Communication

The eMule client and server use the unreliable UDP service for keep-alive and for search enhancements. The amount of UDP packets (note, packets not bytes) generated by an eMule client can reach up to 5% from the total number of packets sent by the eMule client - This depends on the number of servers in the client's server list, the number of sources for each file in the client's download list and the number of searches performed by the user. UDP packets are triggered by a timer which expires every 100 mili-seconds, combining this with the fact that there is a single thread that is responsible for sending UDP traffic results in a maximum rate of 10 UDP packets/second.

3.1 Server keep alive and status information

The client periodically verifies the status of the servers on his server list. This verification is done by using the UDP server status request (see section 6.3.3) and the UDP server description request (see section 6.3.7) messages. The simple keep alive scheme described here generates no more than a few dozen packets per *hour*, In any case the maximum rate of packets is 0.2 packets per second (or a single packet every 5 seconds). When checking the status of a server the client will first send a server status request message and then, only once in every two attempts a server description request as illustrated in the figure 3.1.

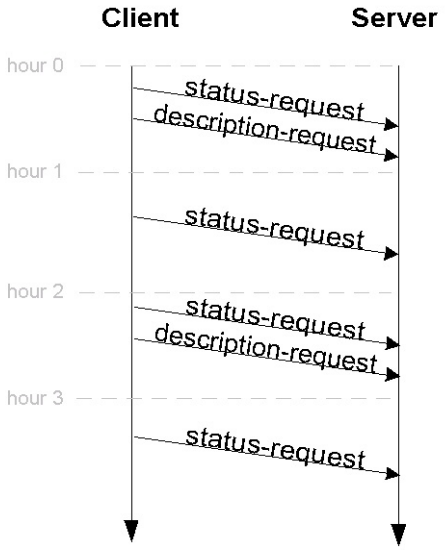

Figure 3.1: UDP Keep alive cycle

The server status request sent by the client includes a random number which is echoed

in the server's reply. In case the number echoed by the server differs from the challenge sent by the client, the information in the reply is discarded. Each time a packet is status request is sent to the server the client advances an attempt-counter. *Any* message from the server (including search results etc) resets the attempt-counter. When the attempt counter reaches a configurable limit the server is considered dead and is removed from the client's server list. Server replies include several data items: The server status reply (section 6.3.4) includes the current number of users and files in the server and also the server's soft and hard limits (section 1.7). The server description reply (section 6.3.8) includes the server name and a short description string. Figure 3.2 illustrates the message flow in the a full keep-alive sequence between a client and an active server.

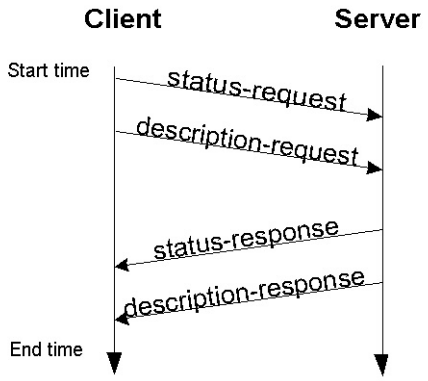

Figure 3.2: UDP Keep alive sequence

3.2 Enhanced file search

The eMule client may be configured to enhance its file searches using UDP. The UDP search request format is almost identical to the TCP search request as described in section **??**. The server responds only in case it has search results. There are two different opcodes of UDP search messages, I couldn't make out the difference between them. The UDP search packets are sent to the servers in the client's server list. See sections 6.3.5 and 6.3.6 for more information.

3.3 Enhanced file-source searches

When the number of sources the client has for a certain file in its download list is less than a configurable limit (100) the client periodically sends UDP get sources packet to servers in his server list to find more sources for the file. A packet may be sent every second, which makes the source searches the most considerable part in the UDP traffic generated by the client. The message format (described in section 6.3.1) is very similar to its TCP counter part. Note that contrary to TCP source searches the UDP source searches are decoupled from file searches and depend only on the number of sources the client has for a given file.

4 Client to Client TCP Communication

After registering to the server and querying it for files and sources the eMule client needs to contact other clients in order to download files. A dedicated TCP connection is created for each [file,client] pair. Connections are closed either when there is no socket activity for a certain period (40 seconds by default) or when the peer has closed the connection.
In order to provide reasonable download rates, eMule doesn't allow a client to start downloading a file until it is possible to provide it (and all other downloading clients) with at least the minimal allowed rate (which is a hard-coded constant currently set to 2.4 kilobytes/second).

4.1 Initial handshake

The initial handshake is symmetric - both parties send the same information to each other. The clients exchange information about each other which includes identification, version and capabilities information. Two types of messages participate - the Hello message (section 6.4.1) and the eMule info message (section 6.5.1) the first is part of eDonkey and is compatible with eDonkey clients, the second is part of the extended client protocol unique to eMule. Figure 4.1 demonstrate a handshake between two eMule clients. Among the things included in the

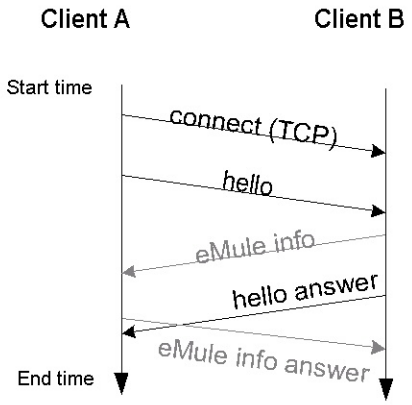

Figure 4.1: eMule client initial handshake

extended information are UDP message exchange, secure identification and source exchange capabilities.

4.2 Secure user identification

Section 1.4 explains shortly about user IDs and the motivation of users to impersonate to other users [3]. The secure user identification is part of the eMule extension. In case the clients support secure identification, it takes place immediately after the initial handshake. The purpose of the secure identification is to prevent user impersonation. When secure identification is applied it takes the following steps:

1. In the initial handshake, B indicates that it supports and wishes to use secure identification
2. A reacts by sending the secure identification message (section 6.5.8) which indicates whether A needs B's public key or not and also contains a 4 byte challenge to be signed by B
3. In case A indicated it needs B's public key then B sends its key to A (section 6.5.9)
4. B send a signature message (section 6.5.10) that is created using the challenge sent and additional double-word which is either A's IP address in case B has low ID or B's ID in case it has high ID. Figure 4.2 illustrates this sequence.

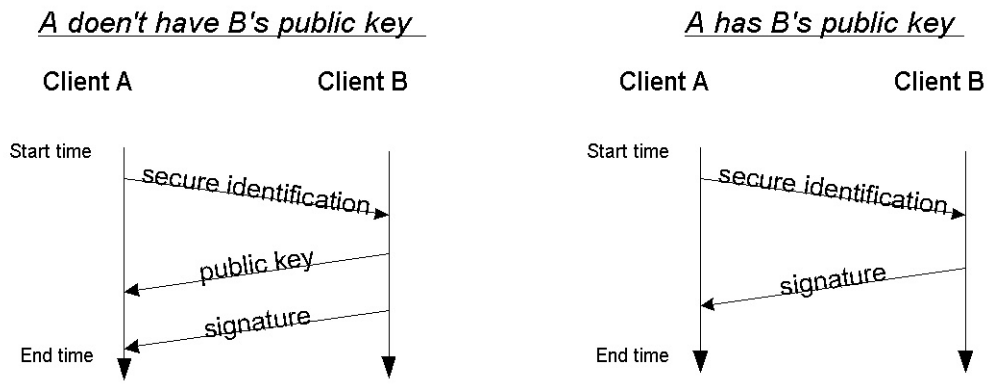

Figure 4.2: Secure identification flow

4.2.1 The credit system

This section briefly describes the client's credit system. The credit system purpose is to encourage users to share files. When a client uploads files to his peer, the downloading client updates his credit according to the amount of data transferred. Note that the credit system is not global - the credit for a transfer is kept locally by the downloading client and will be taken into account only when the uploading client (which earned the credit) will ask to download from this specific client. Credit is calculated as the minimum of:

1. $uploaded\_total * 2 / downloaded\_total$
   When downloaded total is zero this expression evaluates to 10

2. $\sqrt{uploaded\_total + 2}$
   when uploaded total is less then one MB this expression evaluates to 1

The upload/download amounts are calculated in megabytes. In any case the credit can't exceed 10 or be lower than 1.

4.3 Requesting files

As already mentioned a separate connection is created for each [client,file] pair. Immediately after the connection establishment the client sends several query messages regarding the file it wishes to download. A typical, successful scenario is demonstrated in 4.3.

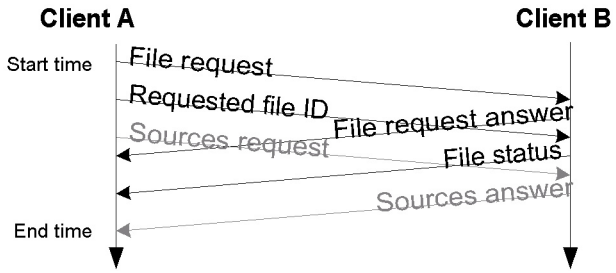

Figure 4.3: File request

4.3.1 Basic message exchange

The basic message exchange is composed from four messages: A sends a file request message (section 6.4.18) immediately followed by a requested file ID message (section 6.4.17). B replies to the file request by a file request answer (section 6.4.15) and to the requested file ID message by a file status (section 6.4.18) message. I couldn't find any reason to divide the information sent in these messages to four messages, it could easily be handled by two messages (a request and a reply).
The extended protocol adds two messages to this sequence a sources request (section 6.5.6) and a sources answer section 6.5.7. This extension is used to pass B's sources (in case B is currently downloading the file) to A. To elaborate, there is no requirement that B completes to download a file before it can send file parts to other clients, B can send to A any part it has completed to download even when it has only a small fragment of the file.

4.3.2 File not found scenario

When A requests a file from B but B doesn't have this file in its shared file list. B skips the file request answer message and send a file not found message (section 6.4.16), immediately after the requested file ID message as demonstrated in figure 4.4.

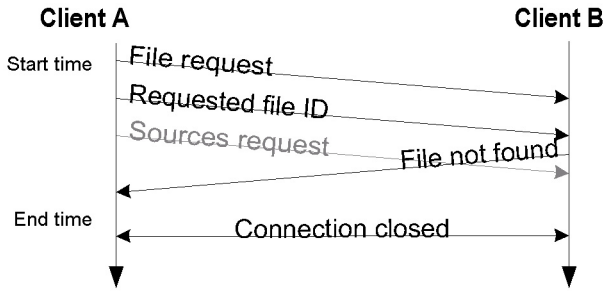

Figure 4.4: File request failure - file not found

4.3.3 Enlisting to the upload queue

In the case where B has the requested file but his upload queue is not empty which means that there are clients that are downloading files and there are probably also clients in the the upload queue A and B perform the full handshake described in figure 4.3 but when A request B to start uploading the file, B adds A to his upload queue and replies with a queue ranking message (section 6.5.4) which contains A's position in B's upload queue. Figure 4.5 illustrates this sequence.

4.3.4 Upload queue management

For each uploaded file the client maintains an upload priority queue. The priority of each client in the queue is calculated on the basis of the client's time in the queue and a priority modifier. At the head of the queue are clients which have the highest **score**. The score is calculated using the following formula: $score = (rating * seconds\_in\_the\_queue)/100$ or ∞ in case the downloading client is defined as a friend. The initial **rating** value is 100 except

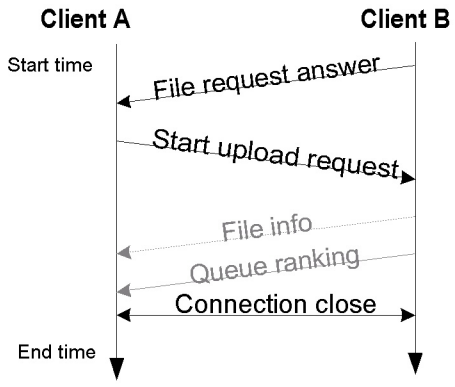

Figure 4.5: File request waiting queue

for banned users which receive 0 rating (and thus are prevented from reaching the top of the queue). The rating is modified either by the downloading client's credit (which ranges from 1 - 10) or by the uploaded file priority (0.2 - 1.8) which is set by the uploading client. When a client's score is higher than the score of the rest of the clients it starts downloading the file. A client will continue to download a file until one of the following conditions occurs:

1. The uploading client was terminated by the user
2. The downloading client has got all the parts it needs for the file
3. The downloading client was preempted by another downloading client which has higher priority than his.

In order to allow a client which just started downloading to get a few megabytes of data before it is preempted, eMule boosts the initial rating of a downloading client to 200 for the first 15 minutes of its download.

4.3.5 Reaching the top of the upload queue

When A reaches the top of B's upload queue, B connects to A, performs the initial handshake and then sends an accept upload request message (section 6.4.11). A can now choose either to continue and download the file by sending a request parts message or to cancel (in case it already got the part from another source) by sending he cancel transfer message (section 6.4.12). Figure 4.6 illustrates these options.

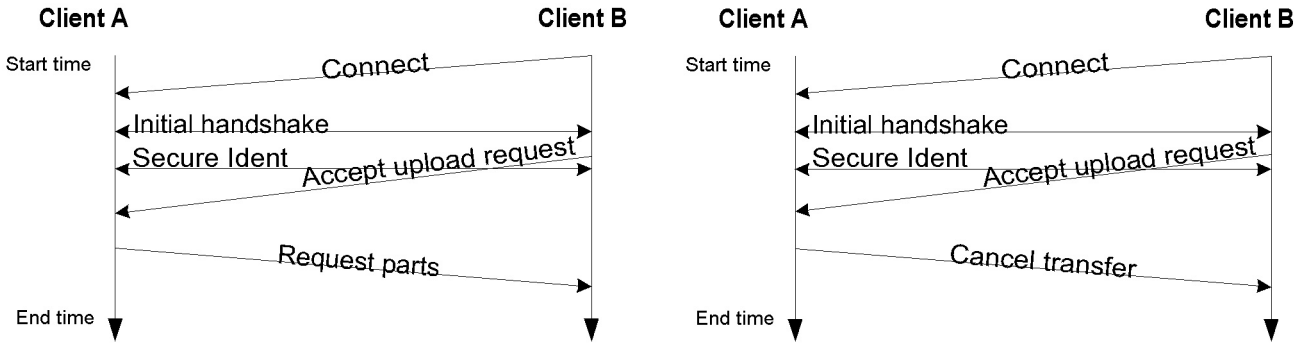

Figure 4.6: File request resume download

4.4 Data transfer

4.4.1 The data packet

Sending and receiving file parts is the major part of eMule's network activity. Allegorizing eMule to FTP would conclude that sending file parts matches the data transactions while all the rest of eMule is the control. A sent file part size can range between 5000 and 15000

bytes (depending also on compression). In order to avoid fragmentation, a file part message is sent in pieces, each piece in a TCP separate packet. In eMule 0.30e the max piece size is 1300 bytes (note that this number relates only to the TCP payload). In other words, while each control messages is sent in a single TCP packet, sometimes shared with other message, data messages are divided to several TCP packets. The first packet contains sending file part message header (section 6.4.3). The rest of the packets contain only data. In case the size of the part sent has a remainder when divided to 1300 it is sent together with the first packet (the one that carries the header). Figure 4.7 illustrates the file part message.

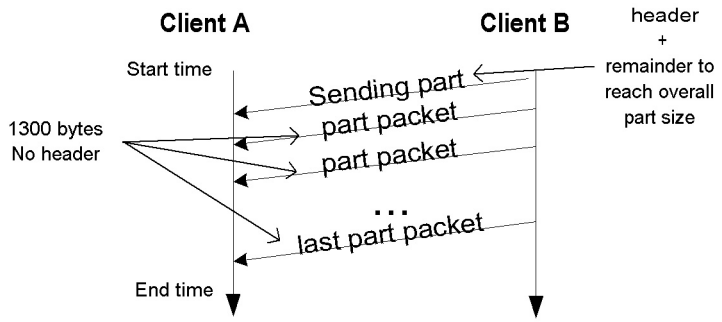

Figure 4.7: File part message details

4.4.2 Data transfer sequence

A part transfer sequence may begin immediately after a file request answer. The downloading client A sends a start upload request (section 6.4.10) which is then replied by an accept upload request message (section 6.4.11). Immediately after that, A starts to request file parts (section 6.4.4) and B replies by sending the requested parts (section 6.4.3). Note that a single file part request may ask for up to 3 parts so each file part request may be replied by up to 3 sending part sequences.

When both clients support the extended client protocol file parts may be sent compressed (section 6.5.3). The extended protocol also supports an optional file info message (section 6.5.5) that may be sent just before the accept upload request message. The part transfer message sequence is illustrated in figure 4.8.

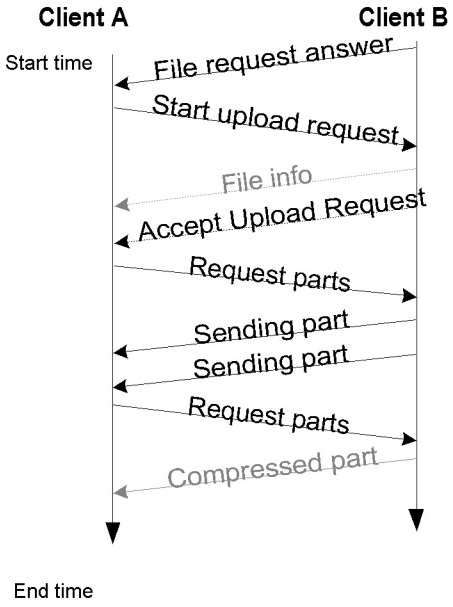

Figure 4.8: File part exchange

4.4.3 Selecting which part to download

eMule selectively selects the download order of parts in order to maximize the overall network throughput and sharing. Each file is divided to 9.28 megabyte parts and each part is divided to 180 KB blocks. The order in which parts are downloaded is determined by the downloading client who sends the request file parts (section 6.4.4) messages. The downloading client may download a single part from each source in any given moment, and all the blocks that are requested from the same source reside in the same part. The following principles apply (in this order) to download part rating:

1. Frequency of the chunk (availability), very rare chunks must be downloaded as quickly as possible to become a new available source.

2. Parts used for preview (first + last chunk), preview or check a file (e.g. movie, mp3)

3. Request state (downloading in process), try to ask each source for another chunk. Spread the requests between all sources.

4. Completion (shortest-to-complete), partially retrieved chunks should be completed before starting to download other one.

The frequency criterion defines three zones: very rare, rare and common. Inside each zone, the criteria has a specific weight, used to calculate the part ratings. Lower rated parts are downloaded first. The following list specifies file rating ranges according to the principles listed above:

- 0-9999 - unrequested and requested very rare parts
- 10000-19999 - unrequested rare and preview parts
- 20000-29999 - unrequested most complete common part
- 30000-39999 - requested rare and preview parts
- 40000-49999 - requested uncompleted common parts

This algorithm usually selects first the rarest parts. However, partially complete parts that are close to completion may also be selected. For common parts, the downloads are spread between the different sources.

4.5 Viewing shared files and folders

There are Two message flows that handle viewing of shared files and folders of peer clients. The first is the view shared files message (section 6.4.21) which is sent immediately after the initial handshake. This message is always replied by a view shared files answer message (section 6.4.22). When the replying client wishes to hide its shared file list the answer will contain zero files (instead of sending a message that signals that the access was denied). 4.9 illustrates the message sequence.

The second message flow starts with a request to view the list of shared folders (section

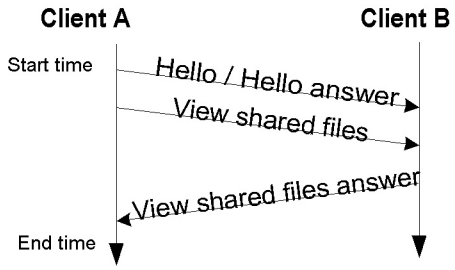

Figure 4.9: View shared files

6.4.23) which is replied by a shared folders list (section 6.4.24) and then, for each folder in the reply a view shared folder content message is sent (section 6.4.25). Each of these messages is replied with a content list (section 6.4.26) when it arrives. Figure 4.10 demonstrates this message sequence.

In case the receiving client is configured to block shared files/folders requests, it replies with a ask shared denied message as illustrated in figure 4.11.

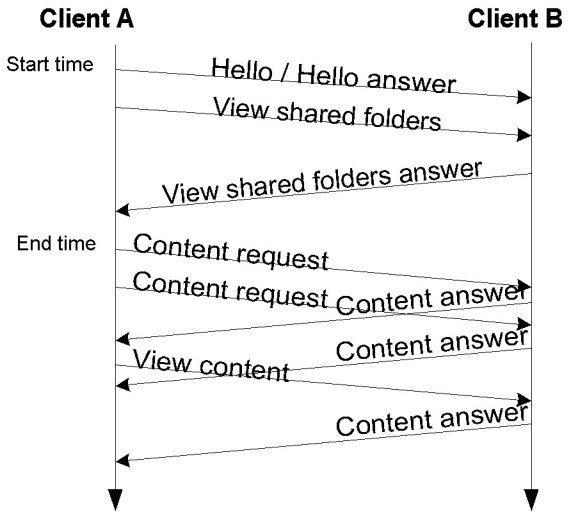

Figure 4.10: View shared file and folders

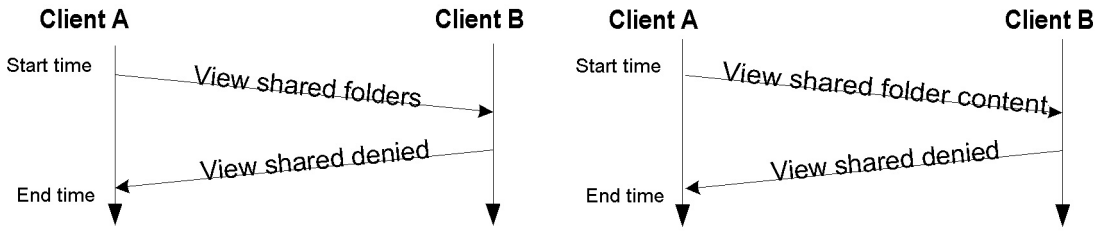

Figure 4.11: View shared denied

4.6 Exchanging part hashsets

In order to get part hashes a hashset request is sent (section 6.4.8) this request is replied by a hashset reply (section 6.4.9) which contains a hashset for each part in the file. Figure 4.12 illustrates this.

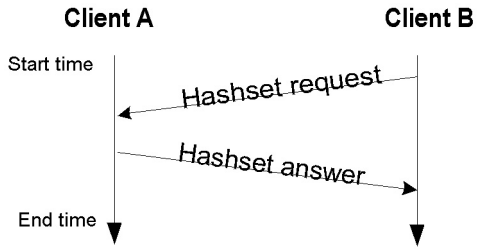

Figure 4.12: Hashset request

4.7 Getting a file's preview

Client's can ask their peer to get a preview of the downloaded file. Previews are application dependant and vary among file types. eMule 0.30e supports only image previews. The message exchange is described in figure 4.13 and contains only two messages: a preview request (section 6.5.11) and preview answer (section 6.5.12)

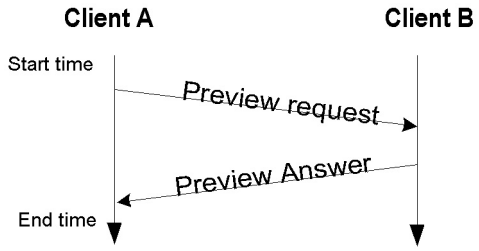

Figure 4.13: Get file preview

5 Client to Client UDP Communication

eMule client periodically sends messages using the UDP protocol. In eMule 0.30e UDP messages are used only for querying the client's position in its peer's download queue. The simple request-response scheme is initiated with a re-ask file message (section 6.6.1). There are three possible replies for this message as demonstrated in figure 5.1:

1. Queue rank - The client's rank in the sender's queue
2. Queue full - The sender's queue is full
3. File not found - The sender doesn't have the requested file in its list

The re-ask file message is sent in intervals of approximately 20 minutes to every client which added the sender to its download queue.

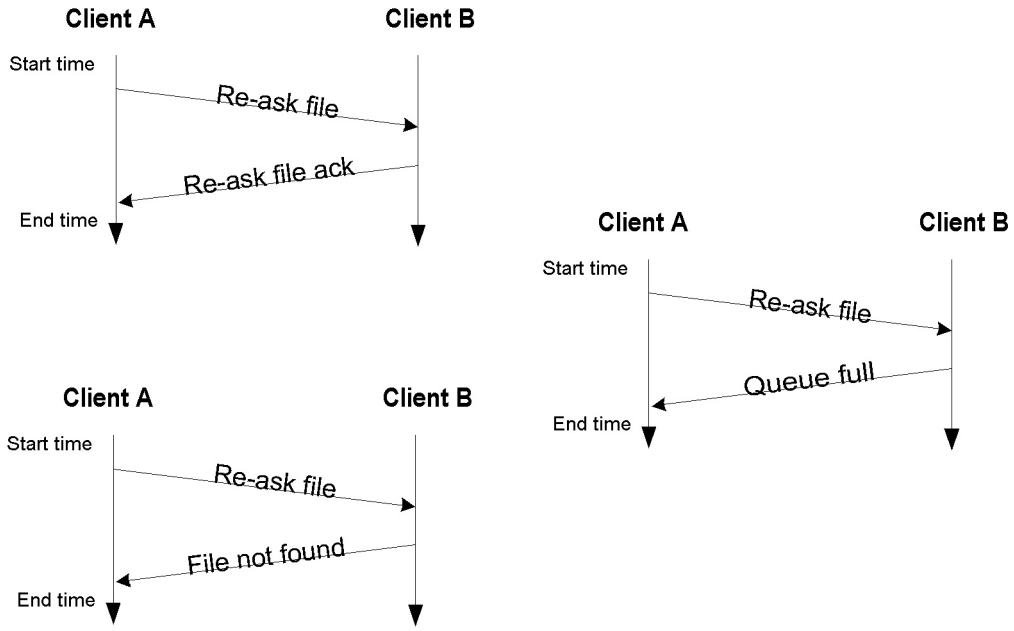

Figure 5.1: Re-ask file message

Bibliography

[1] D. Bickson and D. Malkhi. A study of privacy in file sharing networks. In *Technical Report TR-2003-67 Leibniz Research Center, the Hebrew University of Jerusalem, Israel*, 2003.

[2] eMule at source force. http://sourceforge.net/projects/emule/.

[3] eMule project. http://www.emule-project.net/.

[4] MD4 hash function specification. http://www.ietf.org/rfc/rfc1320.txt.

6 Appendix A - Message Encoding

6.1 General message encoding issues

This section describes the general methodology of encoding messages in TCP/UDP **payload**.

6.1.1 Endianity

All messages are encoded in *little-endian* and not in *big-endian* which is the conventional network byte order. This can be easily explained by the fact the clients/servers are Microsoft Windows-based applications running on Intel processors.

6.1.2 Message Header

All messages have a 6 byte header that has the following structure:

1. protocol - A single byte protocol ID - 0xE3 for eDonkey and 0xC5 for eMule
2. size - 4 byte message size - the message size in bytes not including the header, for example, in case the message doesn't include any payload like in section 6.4.11 then the message length is zero
3. type - A single byte type - a unique message ID

6.1.3 Message Tags

Tags are TLV-like (Type, Length, Value) structures which are used for appending optional data to eMule messages. There are several types of tags, all of them are listed in this section. When referring to specific tags in protocol messages only the tag type is designated, the reader should use this section as a reference to determine the exact structure of a protocol message. Each tag has 4 fields, not all of them serialized into the message:

1. type - 1 byte integer
2. name - could be one of the following:
   - variable length string
   - 1 byte integer
3. value - could be one of the following:
   - 4 byte Integer
   - 4 byte floating point number
   - variable length String.
4. special - 1 byte integer, special tag designator

Tags with Integer values are called Integer tags, similarly we have String and Float tags. The type of a String tag is 2, the type of an Integer tag is 3 and of a Float tag is 4. When tags are encoded on the wire they are encoded in the above order e.g. the **type**. then the **name** and finally the **value**. The type is encoded in a single byte. The name is encoded in a [2 byte] length-value scheme which is used both for String and Integer names. For example the Integer name 0x15 is encoded by the sequence 0x01 0x00 0x15.
Fixed value fields (like Integer and Float numbers) are written as they are, string values are encoded by the same length-value scheme as the name.
Note: The names given to the tags have no special protocol meaning and are here only to ease future reference in protocol message description.

6.2 Client Server TCP Messages

This section describes the messages passed between the server and the client using TCP.

6.2.1 Login

The login message is the first message send by the client to the server after TCP connection establishment. The message length varies as it depends on user configuration such as the user nickname. It is not clear why should the client TCP port be sent (twice!) as it also appears in the TCP header.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x01	The value of the OP_LOGINREQUEST opcode
User Hash	16		Details about user hash can be found in section 1.4
Client ID	4	0	The client ID that is sent on first connection is usually zero. Details about client ID can be found in section 1.3
TCP Port	2	4662	The TCP port used by the client, configurable
Tag Count	4	4	The number of tags following in the message
Name Tag	varies	NA	The user's nickname (configurable in the software). The tag is a string tag and the tag name is an integer of value 0x1
Version Tag	8	0x3C	The eDonkey version supported by the client. The tag is an integer tag and the tag name is an integer of value 0x11
Port Tag	8	4662	The TCP port used by the client. The tag is an integer tag and the tag name is an integer of value 0x0F
Flags Tag	8	0x01	The tag is an integer tag and the tag name is an integer of value 0x20

6.2.2 Server message

Server messages are variable length message that are sent from the server to client on various occasions, the first, immediately after a client login request. A single server-message may contain several messages separated by new line characters ('\r','\n' or both). Messages that start with "server version", "warning", "error" and "[emDynIP: " have special meaning for the client. Other messages are simply displayed to the user.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x38	The value of the OP_SERVERMESSAGE opcode
Size	2	NA	The number of bytes in the remainder of the message not including the fields described so far
Messages	varies	NA	A list of server messages separated by new lines

Special messages

1. version - Usually send in a successful connection handshake
2. error -
3. warning - Usually send when the server denies the connection or when the client has a low ID
4. emDynIP -

6.2.3 ID change

The ID Change message is sent by the server as a response to the login request message and signifies that the server has accepted the client connection. The message size is 14 or 10 bytes depends on sending the optional TCP connection bitmap.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x40	The value of the OP_IDCHANGE opcode
Client ID	4	NA	Details about client ID can be found in section 1.3
TCP connection bitmap	4	0x00000001	Currently only 1 bit (the LSB) has meaning, setting it to 1 signals that the server supports compression

6.2.4 Offer files

This message is used by the client to describe local files available for other clients to download. In case the client has files to offer, the offer-files message is sent immediately after the connection establishment. The message is also transmitted when the client's shared file list changes. Another use of this message is as keep alive, sent periodically to the server. In case the server supports compression file-offers are compressed. When used as keep alive (no files are encapsulated and the message size is small) the message isn't compressed.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x15	The value of the OP_OFFERFILES opcode
File Count	4	NA	The number of files described within. in any case no more than 200. The Server can also set a lower limit to this number
Files	varies	NA	An optional list of files, the format of a single entry is described below.

Single file entry format

The table below describes a single file entry. Several entries may be serialized in a single file-offer message.

Name	Size in bytes	Default Value	Comment
Hash Code	16	NA	The result of a hash (specification TBD) performed on the file contents. The hash is used to uniquely identify files, ignoring name differences between clients
Client ID	4	NA	The client ID in case the client has high ID, or zero otherwise
Client Port	2	0x15	The Client's TCP port or zero in case the client has low ID
Tag Count	4	NA	The number of tags following this field
File Name Tag	varies	NA	The filename (Mandatory). The tag is a string tag the tag name is an integer of value 0x1
File Size Tag	8	NA	The file size in bytes (Mandatory). The tag is an integer tag and the tag name is an integer of value 0x2
File Type Tag	varies	NA	The file type (Optional). One of the following: "Audio", "Video", "Image", "Pro" or "Doc". The tag is a string tag the tag name is an integer of value 0x3
File Format Tag	varies	NA	The file extension converted to lower case (Optional). For example - "zip", "exe". The tag is a string tag the tag name is an integer of value 0x4
Media Length Tag	varies	NA	In case the file is mp3, the song play time (Optional). The tag is a string tag the tag name is the string "length"
Media Bitrate Tag	TBD	NA	In case the file is mp3, the encoding bitrate (Optional). The tag is an integer tag the tag name is the string "bitrate"
Media Codec Tag	varies	NA	In case the file is a movie - the codec used to encode it (Optional, never sent). The tag is a string tag the tag name is the string "codec"

6.2.5 Get list of servers

This message may be sent from the client to the server immediately after a successful handshake completion. The message is sent when the client is configured to expand its list of eMule servers by querying its current server. The message size is 6 bytes.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x14	The value of the OP_SERVERSTATUS opcode

6.2.6 Server status

Sent from the server to the client. The message contains information on the current number of users and files on the server. The information in this message is both stored by the client and also displayed to the user. The message size is 14 bytes.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x34	The value of the OP_SERVERSTATUS opcode
User Count	4	NA	The number of users currently logged in to the server.
File Count	4	NA	The number of files that this server is informed about

6.2.7 List of servers

Sent from the server to the client. The message contains information about additional eMule servers to be used to expand the client's server list. The message size varies (depends on the number of servers transmitted).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x32	The value of the OP_SERVERLIST opcode
Entry Count	1	NA	The number of servers described in this message.
Server entries	(Entry Count)*6	NA	Server descriptor entries each entry size is 6 bytes and contains 4 bytes IP address and then 2 byte TCP port.

6.2.8 Server identification

Sent from the server to the client. Contains a server hash (TBD) ,the server IP address and TCP port (which may be useful when connecting through a proxy) and also server description information. The message size varies.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x41	The value of the OP_SERVERIDENT opcode
Hash	16	NA	A GUID of the server (seems to be used for debug)
Server IP	4	NA	The IP address of the server
Server Port	4	NA	The TCP port on which the server listens
Tag Count	4	NA	The number of tags at the end of the message
Server Name Tag	Varies	NA	The name of the server. The tag is a string tag and the tag name is an integer of value 0x1
Server Description Tag	Varies	NA	A server description string. The tag is a string tag and the tag name is an integer of value 0xB

6.2.9 Search request

Sent from the client to the server. The message is used to search for files by a user's search string. The message size varies. The search string may include the boolean conditions 'AND' 'OR', 'NOT'. The user may specify required file type and size and also set an availability threshold (e.g. show me results that are available from at least 5 other clients)

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x16	The value of the OP_SEARCHREQUEST opcode
Parsed search string	varies	NA	The parsed search string format is described below
File Type Constraint	varies	NA	Optional. A string constraint. The string values are one of ("Audio", "Video", "Pro" or "Image" The type field is 3 bytes: 0x1 0x0 0x3
Min Size Constraint	Varies	NA	Optional. An integer constraint. The file size is provided in mega bytes. The type field is 4 bytes: 0x1 0x1 0x0 0x2
Max Size constraint	Varies	NA	Optional. An integer constraint. The file size is provided in mega bytes. The type field is 4 bytes: 0x2 0x1 0x0 0x2
Availability Constraint	Varies	NA	Optional. An integer constraint. Sets a lower limit on the number of client that poses the searched file The type field is 4 bytes: 0x1 0x1 0x0 0x15
Filename Extension constrain	Varies	NA	Optional. A string constraint. The type field is 3 bytes: 0x1 0x0 0x3

Parsed search string format

The parsed search string encodes a binary expression tree with Boolean operators 'AND', 'OR' and 'NOT' and string operands. The tree is encoded in pre-order . The operators are encoded as 2 byte integers with values of 0x0, 0x100, 0x200 for 'AND', 'OR' and 'NOT' respectively. The strings are encoded in TLV format where the type field is a single byte of value 0x1 and the length field is a 2 byte integer. Note that when the search string is a single word it is encoded as a single string operand (no operators). Later versions of eMule encode a search expression which has only 'AND' operators as a single string, replacing he 'AND's by spaces, this fits the server's search string parsing which fragments a single sentence into a series of words separated by 'AND' operators.

Optional constraints format

The constraints is a sequence of entries. Each entry starts with and 'AND' descriptor (2-byte 0x00) followed by the encoded constraint. Thus, the full search line format is <'search-string' AND constraint1 AND constraint2 etc>, as described in the examples figure below. The encoded constraint is divided to 3 fields:

1. kind - A single byte describing whether this is a string (0x2) or an integer (0x3) constraint.
2. value - Either a type-length encoded string or a 4 byte integer value
3. type - A 3 or 4 bytes describing the constraint's kind (see main table above)

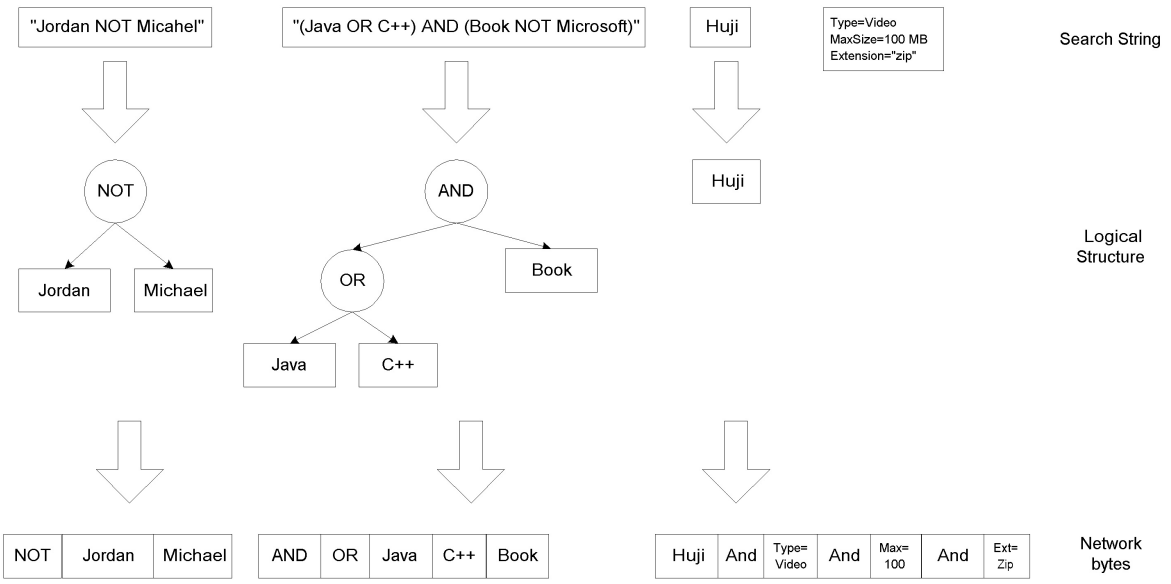

Figure 6.1: Search string encoding example

6.2.10 Search result

A message sent from the server to the client as a reply to a search request. The message is usually compressed. The message size varies.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x16	The value of the OP_SEARCHRESULT opcode
Result Count	4	NA	The number of search results in this message
Result list	Varies	NA	A list of search results

Search result list item format

The table below describes the format of a single search result list-item. Each search result includes A hash that uniquely identifies the file along with details of another eMule client holding the file. There are also several tags describing the file attributes. The tag list

Name	Size in bytes	Default Value	Comment
File Hash	16	NA	A hash value, for unique identification of the file
Client ID	4	NA	Client ID for an eMule peer holding the file
Client Port	2	NA	The TCP port of the client holding the file
Tag Count	4	NA	The number of descriptor tags following
Tag list	Varies	NA	A list of descriptor tags

is described below. Note that most of the tags are optional and that their order is not guaranteed. Tag encoding rules are described is detail an the beginning of this chapter.

Name	Tag name	Tag Type	Comment
File name	Integer, 0x01	String	
File size	Integer 0x02	Integer	
File type	Integer 0x03	String	
File format	Integer 0x04	String	
Sources	Integer 0x15	Integer	The number of available sources for this file
Artist	String "Artist"	String	
Album	String "Album"	String	
Title	String "Title"	String	
Length	String "length"	Integer	
Bitrate	String "bitrate"	Integer	
Codec	String "codec"	Integer	

Table 6.1: search result tag list

6.2.11 Get sources

A message sent from the client to the server requesting sources (other clients) for a file. The message size is 22 bytes.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x19	The value of the OP_GETSOURCES opcode
File hash	16	NA	The requested file hash

6.2.12 Found sources

A message sent from the server to the client with sources (other clients) for a file requested by the client for a file. The message size varies.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x42	The value of the OP_FOUNDSOURCES opcode
File Hash	16	NA	The requested file hash
Sources Count	1	NA	The number of sources in this message
List of sources	Varies	NA	A list of sources

Source list item format

The table below describes the format of a source list-item. Each source includes the details of an eMule client holding the requested file.

Name	Size in bytes	Default Value	Comment
Client ID	4	NA	Client ID for an eMule peer holding the file
Client Port	2	NA	The TCP port of the client holding the file

6.2.13 Callback request

A message sent from the client to the server, requesting another client to call back - e.g. connect to the requesting client. The message is sent by a client that has a high ID who wishes to connect to a low ID client (see section 2.4). The size of the message is 10 bytes.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x1C	The value of the OP_CALLBACKREQUEST opcode
Client ID	4	NA	The ID of the client which is asked to call back

6.2.14 Callback requested

A message sent from the server to the client indicating another client asks the receiving client to connect to it. The message is sent when the receiving client has a low ID (see section 2.4). The size of the message is 12 bytes. The receiving client tries to connect to the IP and port specified by the callback request packet.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x35	The value of the OP_CALLBACKREQUESTED opcode
Client IP	4	NA	The IP of the client which asks to call him back
Client TCP Port	2	NA	The TCP port on which the client listens

6.2.15 Callback failed

A message sent from the server to the client indicating that the client's callback request has failed. The size of the message is 6 (? didn't print one ?) bytes. The receiving client logs the message and discards it.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x36	The value of the OP_CALLBACK_FAILED opcode

6.2.16 Message rejected

A message sent from the server to the client indicating that the server rejected the last command sent by the client. The size of the message is 6 (? didn't print one ?) bytes. The receiving client logs the message and discards it.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x05	The value of the OP_REJECT opcode

6.3 Client Server UDP Messages

This section describes the messages passed between the server and the client using UDP. UDP message headers don't include the message size as it can be deduced from the UDP L3 header. Most of the messages are small, fixed size messages which are sent periodically to the servers on the client's server list.

6.3.1 Get sources

Sent from client to server, requesting sources for a file (other clients that poses the file). This message is sent periodically every second for files that have a low number of sources.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0x9A	The value of the OP_GLOBGETSOURCES opcode
File ID List	NA	NA	A list of file IDs (hashes) (each 16 byte length), The IDs are ordered one after the other without a preceding count

6.3.2 Found sources

Sent from the server to the client as a reply to the UDP get sources message. The message is sent only when the server has sources for the requested file.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0x9B	The value of the OP_GLOBFOUNDSOURCES opcode
File source list	NA	NA	A list of file sources described below

Source list item format

The list includes sources for a single file.

Name	Size in bytes	Default Value	Comment
File ID	16	NA	The file ID to which sources were found
Sources Count	1	NA	number of sources reported therein
List of sources	NA	NA	a list of sources in the same format as the list in the TCP get sources message in section 6.2.11.

6.3.3 Status request

This message is a status request sent every few seconds to the server. The message contains a random 4 byte challenge which should be echoed by the server. The message length is 6 bytes This message is part of the client server UDP keep alive scheme (see section 3.1).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0x96	The value of the OP_GLOBSERVSTATREQ opcode
Challenge	4	NA	A unsigned integer challenge sent to the server, used for reply verification (the corresponding variable is called 'time' on the client)

6.3.4 Status response

The server's response message to the client's UDP status request message. Contains several server information items, note that most of the items are optional.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0x97	The value of the OP_GLOBSERVSTATRES opcode
Challenge	4	NA	Unsigned integer. Could be an echo of the challenge sent by the client or a different number. In the latter case packet processing stops
User Count	4	NA	Optional. The number of users currently logged in to the server
Files count	4	NA	Optional. The number of files in the server's database
Soft files limit	4	NA	Optional. Unsigned integer server soft file limit
Hard files limit	4	NA	Optional. Unsigned integer server hard file limit
UDP flags	4	NA	Optional. The server's UDP flags. Two distinctive flags are defined: 0x01 indicates the the server supports the get sources message. 0x02 indicates the the server support the extended get files message

6.3.5 Search request

This message is sent to servers in the clients list when the client is configured to search using UDP. The message has two optional opcodes (0x98 or 0x92), the latter indicates a later (and enhanced) version of the server. The client decides which opcode to use according to the UDP flags sent by the server in the UDP status response message (section 6.3.4). In order to send the enhanced opcode the server must turn on the 0x02 bit. The UDP flags may also be loaded from a configuration file.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0x98 or 0x92	The value of the OP_GLOBSEARCHREQ or the OP_GLOBSEARCHREQ2 opcodes respectively
Search request parameters	Varies	NA	Same as the search request message parameters see section 6.2.9 in the Client to Server TCP communication section

6.3.6 Search response

A search response message sent for the server to the client. This message is sent as a response to both opcodes in the search message. The message has a format very similar to the TCP search result message although the results are wrapped without a result count. Please read the text in the section describing the TCP search results message to learn more about the various fields of this message (section 6.2.10)

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0x99	The value of the OP_GLOBSEARCHRES opcode
Result list	NA	NA	A list of results (without a count) as described in the section (section 6.2.10)

6.3.7 Server description request

Sent every few seconds to the server. Includes has no payload. Part of the UDP ping scheme 3.1.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0xA2	The value of the OP_SERVER_DESC_REQ opcode

6.3.8 Server description response

Sent from the server to the client as a response to the client's server description request. Contains name and description of the replying server. The message is of variable length.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Type	1	0xA3	The value of the OP_SERVER_DESC_RES opcode
Name	NA	NA	The server's name, a string encoded in a 2-byte length, char array value format
Description	NA	NA	The server's description, a string encoded in a 2-byte length, char array value format

6.4 Client to Client TCP Messages

This section describes the messages passed between clients using TCP. The client to client messages are divided to between eMule and eDonkey types - Note the Header field names in the messages described below. There is also a proprietary type used for the actual passing of passing file parts.

6.4.1 Hello

This message is the first message in the handshake between two e-mule clients. This message is very much like the server login message (see in section 6.2.1). Both messages have the same type code (0x01), they are the only messages in the protocol that have overlapping type code. Both messages provide the same data and even in the same order. There are two main differences: the client hello message begins with a user hash size field while the server login message immediately begins with the user hash value, also the client hello message ends with additional server IP and port information which is not relevant for the server login message.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x01	The value of the OP_HELLO opcode
User Hash size	1	16	The size of the user hash field
User Hash	16		TBD
Client ID	4	0	TBD
TCP Port	2	4662	The TCP port used by the client, configurable
Tag Count	4	4	The number of tags following in the message
Tag list	varies	NA	A list of tags specifying remote client's properties
Server IP	4	NA	The IP of the server to which the client is connected
Server TCP Port	2	NA	The TCP port on which the server listens

There are three types of tags that may appear in the tag-list. The port tag is optional and usually not provided. Tag encoding rules are described is detail an the beginning of this chapter.

Name	Tag name	Tag Type	Comment
Username	Integer, 0x01	String	
Version	Integer 0x11	String	
Port	Integer 0x0F	Integer	

6.4.2 Hello answer

Sent as an answer to a Hello message. Contains exactly the same fields as the hello message except for the message type.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x4C	The value of the OP_HELLOANSWER opcode
Hello fields			The same fields as in the hello message starting with the user hash

6.4.3 Sending file part

This message contains a part of a downloaded file. The message size (in the header) indicates the overall part size and not only the size of the packet in which this message is sent. This message is divided to several data packets each with payload smaller than the maximum TCP MTU - in eMule 0.30e the payload size is 1300 bytes. Section **??** discusses sending file parts in detail. See also section 6.5.3 for details about sending compressed file parts.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the *sent part* in bytes not including the header and size fields
Type	1	0x46	The value of the OP_SENDINGPART opcode
File ID	16	NA	A unique file ID calculated by hashing the file's data
Start Pos	4	NA	The start position of the downloaded data
End Pos	4	NA	The end position of the downloaded data
Data	NA	NA	The actual downloaded data. This data may be compressed.

6.4.4 Request file parts

Send to a peer client to request file parts. The message may request a max number of 3 file parts to download.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x47	The value of the OP_REQUESTPARTS opcode
File ID	16	NA	A unique file ID calculated by hashing the file's data
Part 1 Start offset	4	NA	Start offset of the part 1 in the file
Part 2 Start offset	4	NA	
Part 3 Start offset	4	NA	
Part 1 End offset	4	NA	End offset of the part 1 in the file
Part 2 End offset	4	NA	
Part 3 End offset	4	NA	

6.4.5 End of download

Used by a downloading client to indicates a file download is complete. This message is never sent by eMule 0.30e but the client handles receiving this message (by freeing related resources).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x49	The value of the OP_END_OF_DOWNLOAD opcode
File ID	16	NA	The file ID

6.4.6 Change client ID

Sent when the client has changed servers (e.g. disconnected from his current server and connected to a new server) and as a result got a new client ID.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x4D	The value of the OP_CHANGE_CLIENT_ID opcode
Client ID	4	0	The new client ID
Server IP	4	NA	The IP of the server to which the client is connected
Server TCP Port	2	NA	The TCP port on which the server listens

6.4.7 Chat message

Used to implement the chat services provided by eMule.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x4E	The value of the OP_MESSAGE opcode
Length	2	NA	The length of the message
Message	Varies	NA	The actual message

6.4.8 Part hashset request

Send as a request for the hash of all the requested file and for each and every part in the file. The unique file ID and the file hash are explained in section 1.5.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x51	The value of the OP_HASHSETREQUEST opcode
File ID	16	NA	The file ID of the requested file

6.4.9 Part hashset reply

A reply to a file hash set request containing the global hash (for all the file) and a hash for every part of the file. When the processing of this message is complete the client sends a start upload request.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x52	The value of the OP_HASHSETANSWER opcode
File Hash	16	NA	The hash extracted from all the file
Part Count	2	NA	The number of parts in the file
Part Hashes	Varies	NA	A hash for each file part - the size of each hash is 16 bytes

6.4.10 Start upload request

A start upload request message. This message starts the file download sequence which is discussed in section 4.3.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x54	The value of the OP_STARTUPLOADREQ opcode
File ID	16	NA	The ID of the requested file

6.4.11 Accept upload request

An ack, indicating the upload request was accepted and the uploading client is now waiting for part requests. The message has no fields except the standard eMule message header. This message is part of the part sending protocol, more details in section **??**.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4	0	The size of the message in bytes not including the header and size fields
Type	1	0x55	The value of the OP_ACCEPTUPLOADREQ opcode

6.4.12 Cancel transfer

A request to cancel a file transfer. The request doesn't contain any fields. The typical scenario in which this message is sent is when a downloading's has reached the top of the queue but it has already downloaded the file from another source.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4	0	The size of the message in bytes not including the header and size fields
Type	1	0x56	The value of the OP_CANCELTRANSFER opcode

6.4.13 Out of part requests

Not used in eMule 0.30e. Indicates that a downloading clients has completed to download all the parts sent by the uploading client and is waiting for more. Note that no actual action is taken by the receiving client.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4	0	The size of the message in bytes not including the header and size fields
Type	1	0x57	The value of the OP_OUTOFPARTREQS opcode

6.4.14 File request

A message sent from a client requesting a file from another client. The message contains the ID of the requested file and a status field describing which parts are already downloaded.

The part status field

eMule allows clients to download file parts from other clients even when the providing client has not yet completed downloading the requested file. the *Part status* field helps distinguishing between a file that simply doesn't exist on the requested client and a file that is only partially downloaded on by it. In case the file doesn't exist then the value of *Part status* is 0. In case the file is partially downloaded, the first 2 bytes are an integer giving the number of parts already downloaded and the last byte is a bitmap which indicates which eighth of the file is completely downloaded (by setting the matching bit to 1).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x58	The value of the OP_FILEREQUEST opcode
File ID	16	NA	Unique file ID
Part Status	3	NA	Optional, sent if the extended request version indicated in the eMule info message is greater than zero. The file significance is explained in this section
Source count	2	NA	Optional, sent if the extended request version indicated in the eMule info message is greater than one. Indicated the current number of sources for this file

6.4.15 File request answer

A file request answer, sent as a reply to a file request message. This message is only one of the possible replies to a file request more details in section 4.3

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x59	The value of the OP_FILEREQANSWER opcode
File ID	16	NA	A unique file ID (hash)
Name length	2	NA	The filename length
Filename	Varies	NA	The filename (in the length specified)

6.4.16 File not found

This message replies a file request and indicates that a requested file or part of a file was not found on the client.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x48	The value of the OP_FILEREQANSNOFIL opcode
File ID	16	NA	The non existent file ID
Part Status	3	NA	Explained in file request (section 6.4.14)

6.4.17 Requested file ID

A file download request message. The message only specifies the File ID.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x4E	The value of the OP_SETREQFILEID opcode
File ID	16	NA	The ID of the requested file

6.4.18 File status

Sent as a reply to a requested file ID message in the case the client has the requested file.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x50	The value of the OP_FILESTATUS opcode
File ID	16	NA	The ID of the file whose status is reported
Part Status	3	NA	Explained in file request (section 6.4.14)

6.4.19 Change slot

Not sent by eMule 0.30e. Ignored when received.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4	0	The size of the message in bytes not including the header and size fields
Type	1	0x5B	The value of the OP_CHANGE_SLOT opcode

6.4.20 Queue rank

Not sent by eMule 0.30e. Ignored when received. For the extended protocol queue ranking message see section 6.5.4.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x5C	The value of the OP_QUEUERANK opcode

6.4.21 View shared files

Request for a list of shared files. This request is answered only in case the requesting client is not banned and the isn't configured to ignore such requests.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x4A	The value of the OP_ASKSHAREDFILES opcode

6.4.22 View shared files answer

This message encodes the shared files for a requesting client. The message contains a list of shared file details.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x4B	The OP_ASKSHAREDFILESANSWER opcode
Result Count	4	NA	The number of filenames encoded in this message
Result List	Varies	NA	Exactly the same encoding as in a server's search result (section 6.2.10)

6.4.23 View shared folders

Request for a list of shared directories. This request is answered only in case the requesting client is not banned and the isn't configured to ignore such requests.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x5D	The value of the OP_ASKSHAREDDIRS opcode

6.4.24 View shared folders answer

The reply to the view shared folders request. The reply includes a list of shared folders.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x5F	The value of the OP_ASKSHAREDDIRSANS opcode
Folder count	4	NA	The number of folders described by the message
Folder list	Varies	NA	A list of folder description in a length value format described below

Folder list format

The folder list format is a sequence of entries, each entry consists of a short integer (uint16) length and a string value.

6.4.25 View content of a shared folder

A message that asks to list the contents of one of the client's shared directories. The request is respected only in case the directory is shared and the requesting client is a trusted client.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x5E	The value of the OP_ASKSHAREDFILESDIR opcode
Name length	2	NA	The length of the directory name that follows
Directory name	Varies	NA	A shared folder name

6.4.26 View shared folder content answer

The reply to the shared files directory request (OP_ASKSHAREDFILESDIR). The reply includes a list of filenames (all the files that exist in the shared directory).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x60	The value of the OP_ASKSHAREDFILESDIRANS opcode
Name length	2	NA	The length of the directory name that follows
Directory name	Varies	NA	A shared folder name (which the file list below refers to)
Files Count	4	NA	The number of file names following this field
Files list	Varies	NA	A list of file names in (unit16-length, string-value) format

6.4.27 View shared folder or content denied

Sent to signify that a shared files or directory request has been denied. This message is sent to deny both OP_ASKSHAREDDIRS and OP_ASKSHAREDFILESDIR requests.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xE3	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x61	The value of the OP_ASKSHAREDDENIEDANS opcode

6.5 Client to Client TCP extended messages

The messages below are part of eMule and are not part of eDonkey, they are called extended protocol messages.

6.5.1 eMule info

A message carrying general information about the eMule client. The tag list below is compliant with eMule 0.30e and might differ in later or previous versions of the protocol. This message is immediately replied by an eMule info answer message (section 6.5.2). The message is part of the client initial handshake (section 4.1). eMule info contains several tags which indicate the eMule client's capabilities the tag significance and value range is briefly explained below.

UDP Version tag

Provides the UDP protocol version of this client - e.g. which client to client UDP messages are supported. The tag is an integer tag an its name is 0x22. In 0.30e the tag value is 0x03

Source exchange tag

The tag indicates whether the client supports exchanging sources with other clients, the tag value is treated as the source exchange support version. The tag is an integer tag an its name is 0x23. In 0.30e the tag value is 0x02 and the allowed values are either 0x01 or 0x02

Comments Support tag

Indicates whether file comments are supported. The tag is an integer tag an its name is 0x24. In 0.30e the tag value is 0x01 which indicates that the client supports comments.

Extended requests tag

The tag indicates the client's ability to send or receive extended file requests. The tag is an integer tag an its name is 0x25. In 0.30e the tag value is 0x02.

Extended option tag

In eMule0.30e the tag provides information regarding two features - the lower 6 bits indicate whether the client supports secure identification, the next bit (the 7th) indicates whether the client supports file preview. The tag is an integer tag an its name is 0x27. The values of the

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x01	The value of the OP_EMULEINFO opcode
Client Version	1	NA	The version of the eMule client
Protocol Version	1	0x01	The eMule protocol version
Tag Count	4	7	The number of tags following this
Compression tag	8	NA	Designates that this client supports compression. The tag is an integer tag an its name is 0x20. In 0.30e the tag value is 0x01
UDP Version tag	8	NA	Explained in the text
UDP Port tag	8	NA	Provides the UDP port on which this client listens. The tag in an integer tag an its name is 0x21. The default value is 4672
Source exchange tag	8	NA	Explained in the text
Comments tag	8	NA	Explained in the text
Extend requests tag	8	NA	Explained in the text
Extended options tag	8	NA	Explained in the text

6.5.2 eMule info answer

Sent as a reply to an eMule info message. Contains exactly the same information. The message is part of the client initial handshake (section 4.1).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x02	The value of the OP_EMULEINFOANSWER opcode
eMule Info fields			This message has the same fields as an eMule info message.

6.5.3 Sending compressed file part

This message is used to send compressed file parts. A part may be compressed, the compressed content is sent in several consequent packets just like the ordinary sending part message (section 6.4.3).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x40	The value of the OP_COMPRESSEDPART opcode
File ID	16	NA	A unique file ID
Part start offset	4	NA	The start offset of the sent part
Compressed content size	4	NA	The size of the compressed content
Compressed content	Varies	NA	Compressed part

6.5.4 Queue ranking

A report to a peer client on its queue ranking - its waiting position in the download queue. The message is usually sent when a client is added to the waiting queue for a certain file. In most cases, this message if followed by closing the client's TCP connection.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x60	The value of the OP_COMPRESSEDPART opcode
Queue position	2	NA	The position of the client in the queue
Buffer	10	0	10 zero bytes, purpose unknown

6.5.5 File info

A message that contains file description for a specific file. Sent as a reply for a file request or a file upload request message . The message doesn't contain the file ID which is implicitly extracted by the receiving client under the assumption that a single TCP connection works on a single file.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x61	The value of the OP_FILEDESC opcode
File rate	1	NA	A byte providing some kind of rating on the file
Comment length	2	NA	the length of the comment that follows (no longer than 128)
Comment	Varies	NA	The comment itself

6.5.6 Sources request

Sent to request sources (other client that have a required file).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x81	The value of the OP_REQUESTSOURCES opcode
File ID	16	NA	The file ID of the required file

6.5.7 Sources answer

Sent as a reply to a request sources message. This message may be compressed (when its uncompressed size is larger than 394 bytes).

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x82	The value of the OP_ANSWERSOURCES opcode
File ID	16	NA	The file ID on which the answer is sent
Source Count	2	30	Always zero in eMule 0.30e
List of sources	Varies	NA	A list of sources for the required file as described below

Source list item format

The table below describes the format of a source list-item. Each source includes the details of an eMule client holding the requested file and the server to which this was last seen connected.

Name	Size in bytes	Default Value	Comment
Client ID	4	NA	Client ID for an eMule peer holding the file
Client Port	2	NA	The TCP port of the client holding the file
Server IP	4	NA	The IP address of the server from which this user was found
Server Port	2	NA	The TCP port on which the server listens to client connections
Client hash	16	NA	**OPTIONAL** - sent only if both clients have a source exchange version see (OP_EMULEINFO message) larger than 1

6.5.8 Secure identification

Sent usually after the initial client handshake (section 4.1). The message is sent only in case the peer client supports secure identification (as indicated in the eMule info message). The message indicates whether the sending client has the peer's public key and also provides a random value as a challenge for the peer client to sign.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x87	The value of the OP_SECIDENTSTATE opcode
Operation	1	2	Indicates whether a public key and a signature are needed (2) or that only a signature is needed (1)
Challenge	4	NA	A random word for the peer to sign

6.5.9 Public key

Contains a the client's public key to be used when both sides support encryption.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x85	The value of the OP_PUBLICKEY opcode
Public key length	1	76	The length of the key
Public key	Varies	NA	Usually 76 bytes length

6.5.10 Signature

The client signs a 4 byte challenge using its public key. The challenge is composed from a random word sent by the remote client and another word - either the remote client's IP address (in case the signing client has a low ID) or the signing client's ID in case the signing client has a high ID.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x86	The value of the OP_SIGNATURE opcode
Signature length	1	48	The length of the signature
Signature	Varies	NA	Usually 48 bytes length

6.5.11 Preview request

Request an image preview for a specific image file.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x90	The value of the OP_REQUESTPREVIEW opcode
File ID	16	NA	The file ID

6.5.12 Preview answer

An image preview answer message. Contains a preview of the requested image file or just the file ID if the requested file is not an image.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x91	The value of the OP_PREVIEWANSWER opcode
File ID	16	NA	The file ID or zero, in case only the image is sent
Frame count	1	NA	The number of frames sent for this image
Frames	Varies	NA	The image frames encoded in a (4 byte) length, value encoding

6.6 Client to Client UDP Messages

This section describes the messages passed between the server and the client using UDP. All the messages sent using UDP are part of the eMule extension and thus the protocol ID is the eMule protocol and not eDonkey.

6.6.1 Re-ask file

Re ask a peer client about a status of a requested file. The message is used when the asking client isn't in the requested client's queue and wants to periodically find out whether it can start download the file. This message has three possible responses which are documented below.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x90	The value of the OP_REASKFILEPING opcode
File ID	16	NA	The ID of the file which is reasked
Source count	2	NA	Optional. Unsigned integer, The current number of sources of the requested file

6.6.2 Re-ask file ack

One of the responses to a re-ask file request indicating the the peer client has received the request, there is vacancy in its download queue and the requesting client is in the queue with the specified rank.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x91	The value of the OP_REASKACK opcode
Rank	2	NA	Unsigned integer. The requesting client's position in the requested client's queue

One of the responses to a re-ask file ping request indicating that a file wasn't found, the message doesn't indicate which file was not found.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x92	The value of the OP_FILENOTFOUND opcode

6.6.3 Queue full

One of the responses to a re-ask file request, the message doesn't indicate which file queue is full.

Name	Size in bytes	Default Value	Comment
Protocol	1	0xC5	
Size	4		The size of the message in bytes not including the header and size fields
Type	1	0x93	The value of the OP_QUEUEFULL opcode