Request For Comments

May 2020

# THE LIRC PROTOCOL (REVISION 1)

## Status of this Memo

This RFC specifies a students project protocol for implementing a chat service using two modes of communication.
The first is a broadcast mode, which delivers to all connected clients a message sended by a client.
And the second one is like a unicast mode between two clients. Clients using this mode can send private messages or files to each other without requiring being relayed by the server.

## Summary

LIRC stands for Legal Internet Relay Chat.

LIRC is a protocol used to facilitate communication in the form of text. The chat process works on a client/server networking model. However, unlike with IRC, this protocol allows clients to exchange files and private messages between them. In this case, the server is only here to exchange addresses between clients.

This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

## Acknowledgements

The protocol was originally designed by two students, Judicaël KOFFI and Christelle NGUYEN.

## 1. Purpose

LIRC is a protocol to communicate in the form of text, and is implemented on top of the Transmission Control Protocol (TCP).
It allows a client to connect to a server and send messages to all connected clients on this server. LIRC also allows direct communication with another connected client, without going through the server.
A private connection between two clients is mandatory as the client asks the server to negotiate the access to target client information, before performing a direct connection with the client.

## 2. Overview of the protocol

Any access to chat service begins with a connection request in anonymous (only login is mandatory) or authentication mode with a login and a password from a client to the server. The login has to be unique. Any message used in this protocol is encoded with standard charset UTF-8.

When the authentication is done, the client can decide to chat either with all connected clients or, only with a specific client privately. He can also send files to a client in particular.

All requests sent to the server by a client or client to client in private connection must be acknowledged by an acknowledgment packet.

The client can disconnect from the chat too. To be disconnected, the client must send a disconnection request to the server.

## 3. Relation to other Protocols

LIRC protocol is designed to be implemented on top of Transmission Control Protocol (TCP). TCP is a connection oriented protocol, so it is assured some verifications and assurance for exchanges occurred in the network.

TCP guarantees that all bytes received will be identical and in the same order as those sent.

Please refer to this link, to learn more about TCP: https://tools.ietf.org/html/rfc793

## 4. Initial Connection Protocol

To establish a connection with the server, a client must send a request of connection.
There are two types of connection request:

- an anonymous connection where a password is not mandatory, only the login is
- an authenticated connection where both password and login are mandatory

After sending the request, the client can receive an acknowledgment (ACK) packet from the server, containing a byte representing an opcode 2 which indicates success response from server.
Clients can also receive error packets from the server where the first byte contains opcode -1 which indicates an error occurred during connection.
All packets with opcode -1 contain a string encoded in UTF-8 to give a description of the error.

Then, once the connection is established, the client can do the following things :

A. For exchanges between CLIENT and SERVER

- To send a message to all the clients connected to the server (as described above), like a broadcast message. Then, the server sends back an Ack message, to acknowledge the request.

- To send a request to perform a private connection to another client (let us say client A to client B). The server then has to relay the request of connection to client B and if client B accepts a private connection request, the server sends the address of client B(IP and port) to client A, and client A can perform private connection to client B.

B. For exchanges between CLIENT and CLIENT:

When the connection is established, the client is able to :

- Send private message
- Send file

All exchanges packet structure will be detailed in the following part.

**5. LIRC Packets**

LIRC supports 16 types of packets, all of which have been mentioned above.

| opcode | operation |
|---|---|
| 0 | Client requests to server an anonymous connection |
| 1 | Client requests to server an authenticated connection |
| 2 | Server acknowledgment for client request |
| 3 | Client sends a message to server to be broadcasted for all connected clients |
| 4 | Server sends broadcast message to a client |
| 5 | Client requests server to have a private connection with another client |
| 6 | Server notify a targeted client for incoming private connection to him |
| 7 | Client positive response to server for an incoming private connection request |
| 8 | Server positive response to sender client for private connection request |
| 9 | Client private message send |
| 10 | Client ack packet |
| 11 | Client private file send |
| 12 | Client requests for disconnection to server / client |
| -1 | Server error packet |
| -3 | Server negative response to sender client for private connection request |
| -4 | Client negative response to server for incoming private connection request |

**Opcode 0**:

A client connection request without password has the following packet structure:

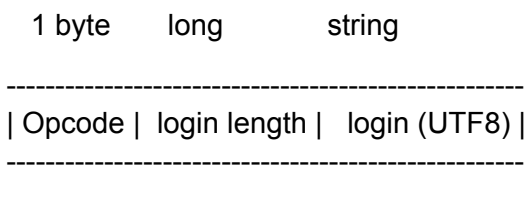


Figure 5-1: anonymous connection request packet

Anonymous connections packets (opcode 0) have the format shown in Figure 5-1.

- login size: a long number in big endian used to represent the length of login byte sequence encoded in UTF-8.
- login: a string encoded in UTF-8 used to identify a client in chat

**Opcode 1:**

A client connection request with password packet have following structure:

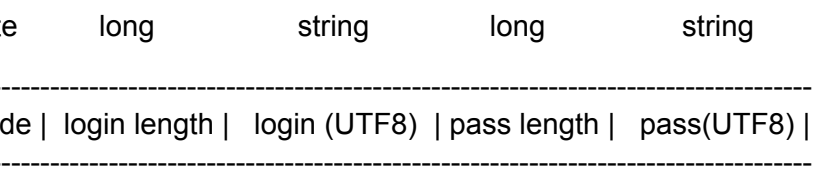


Figure 5-2: authenticated connection request packet

Authenticated connections packets (opcode 1) have the format shown in Figure 5-2.

- The login size is a long number in big endian used to represent the length of login byte sequence.
- The login is a string encoded with UTF-8 charset.
- The pass size is a long number in big endian used to represent the length of password character sequence.
- The pass is a string encoded in UTF-8.

**Opcode 2 :**

All requests sent to the server must be acknowledged by sending back to sender a packet (figure 5-3 below) with opcode 2, which represents a good reception of request :

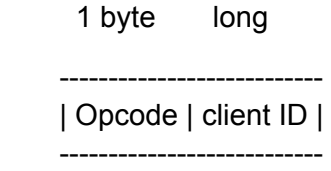


Figure 5-3: server ack packet

A clientID is a long value generated by a server to identify each connected client. This value is communicated to the client after a successful connection.
Each client must provide this identifier in all of his following request packets after being connected to the server.

**Opcode 3 :**

Concerning broadcast messages, all these requests have to be sent to the server. This one is in charge of distributing the message to all the clients connected to him. The broadcasting request is determined by the opcode 3 and have the following structure :

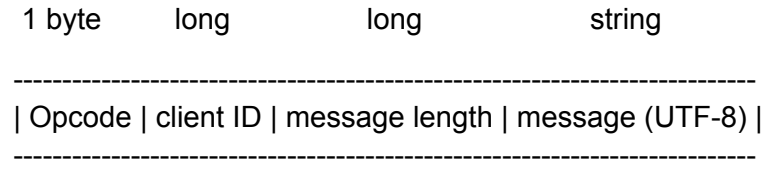


Figure 5-4 : message to be broadcasted from client to server packet

- The client ID is a long value in big endian representing the value a server associates to a current client (which is here the sender client)
- The message length is the size of the entire message (following element) as a long in big endian.
- And the message, encoded in UTF-8, represents the content itself in string.

**Opcode 4:**

Once the server has acknowledged the broadcast message packet from the client, he will send to this client a ACK packet to assure the client that the request has been received (cf. figure 5-3).

After sending an ACK packet to a client, the server sends to all connected clients a packet containing a previous received message. This packet starts with the opcode 4 and has the following structure:

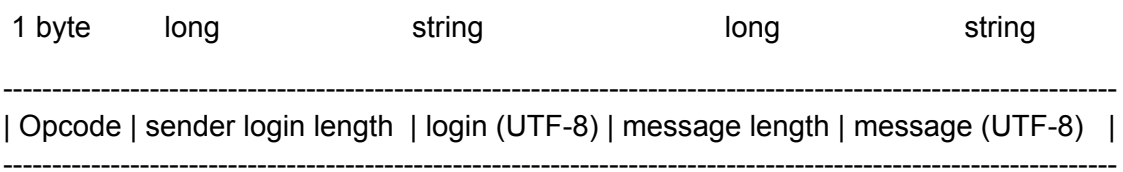


Figure 5-5 : message broadcasted from server to clients packet

- The sender login length is a long value in big endian which represents how many bytes must be read to have the author login encoded in UTF-8.
- The sender login length is followed by a string representing the login encoded in UTF-8.
- The two last parts of the packet are another long value in big endian representing the length of the following message in long and after we have the message encoded in UTF-8.

After receiving any message from the server, the client must send back acknowledgement to the server. A client acknowledgement packet structure can be found on figure 5-11.

**Opcode 5 :**

When a client wants to have a private connection with another client, he must send a private request packet, with opcode 5, to the server. The server will send to a targeted client for a private connection, a packet to indicate that a client wants a private connection with him. This type of request is illustrated like this:

| 1 byte | long | long | string |
|---|---|---|---|
| Opcode | client ID | target client login length | target client login (UTF-8) |

Figure 5-6: request private connection to the server packet

- The client ID is a long and in big endian. It represents the sender client's identifier requesting to the server a private connection to another client.
- The target login length is the size of the receiver client's login encoded in UTF-8. The value is a long in big endian.
- And the last element login is simply the receiver client's login itself in string, encoded in UTF-8.

**Opcode 6 :**

To notify the targeted client for an incoming private connection request (from another connected client), the server send the following packet with the opcode 6, to ask if the target wants to accept a private connection :

| 1 byte | long | long | string |
|---|---|---|---|
| Opcode | target client ID | sender client login length | sender client login (UTF-8) |

Figure 5-7: request incoming private connection to target client packet

- The clientID is the client's identifier which wants the server to send the request. It is a long and in big endian.
- The sender client login length is the size of the sender's login. It is a long and in big endian. The size is encoded in UTF-8.
- And the sender client login is the sender's login itself, encoded in UTF-8, in string type.

**Opcode 7:**

After being notified by the server of an incoming private connection with him, if a client accepts a private connection, he must send an approval packet to the server.

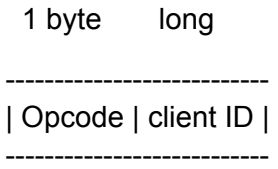


Figure 5-8: client positive response packet

The content is an opcode with value 7, and a long value in big endian which represents the current client Id.

**Opcode 8:**

In addition to this request of private connection to another client, the server has to respond back to this client with a packet. But, the response can be positive (opcode 8) or negative (opcode -3, see error opcode later).

If the response given by the server is positive (the receiver has accepted the connection), it has the structure below with the opcode 8:

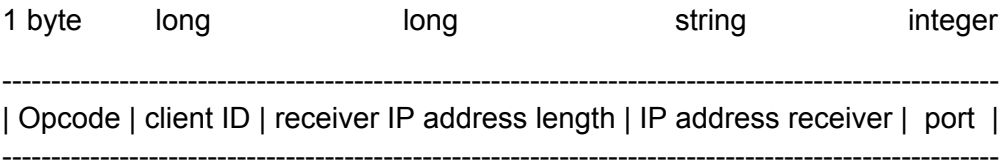


Figure 5-9 : server positive response to request private connection packet

- The clientID is in long and in big endian. It represents the current client's identifier.
- The receiver IP address length is a long which represents the length (encoded in UTF-8) of the client's IP address we want to chat with. The length is necessary to get the right number of byte to decode the IP address.
- Then the IP address receiver is the IP address itself. It is in string and encoded in UTF-8.
- The last part of this packet is an integer which represents the receiver port value

After getting all these values, a client is able to perform a direct connection with another client.

**Opcode 9 :**

Once the private connection is established between two clients connected to a server, they can chat with each other, and so send messages. The packet, which starts by opcode 9, for exchanging messages has the structure below:

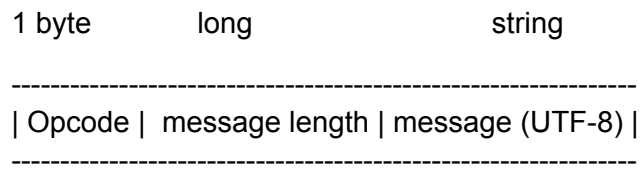


Figure 5-10 : client sending private message packet

- The message length is the size of the whole message sent, encoded UTF-8. It is represented by a long type.
- The message corresponds to the content of the message itself, encoded in UTF-8 and is a String.

**Opcode 10 :**

The following packet is a packet used by a client to acknowledge all received messages from the server or another client. The opcode is 10.

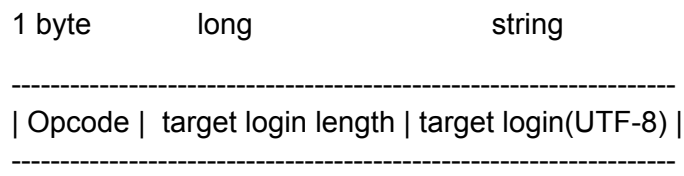


Figure 5-11 : client sending ACK packet

- The target login length is the client login's size sending the ack. It is a long, in big endian and encoded in UTF-8.
- And the target login is the client's login who sent a message first, and who is waiting for an ACK. This login is a String, and is encoded in UTF-8.

**Opcode 11 :**

Once the private connection is established between two clients connected to a server, they can send files to each other. The opcode used for file sending is 10. And he packet for exchanging files can be illustrated like this :

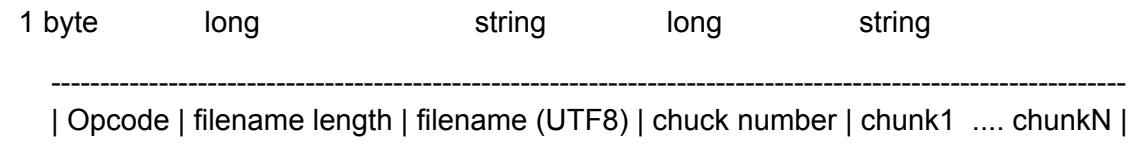


Figure 5-12 : client send file packet

- After opcode, there is a long value in big endian which represents the filename's size (encoded in UTF-8)
- Filename is a string which is the filename encoded in UTF-8
- Number of chuck is in big endian, long, to indicate how many times the file content was subdivided
- chunk1 … chunkN are file content in String

The size of each chunk is a static value fixed to 4096 bytes .

**Opcode 12 :**

When a client wants to disconnect from the server or a client, he has to send a request to this one with the following packet structure (opcode 12) :

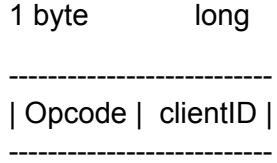


Figure 5-13 : client sending disconnection packet

- The clientID is the client's identifier who wants to be disconnected from the server or a client

**Errors opcodes:**

All opcodes described on the previous part concern the packet in the nominal case, but in the following part we will describe all opcodes which indicated an error which occured, like a wrong password, unavailable login etc.

**Opcode -1:**

In case of connection error, a server sends a packet, where the opcode is -1, with the following structure (figure 5-14) to indicate a type of error :

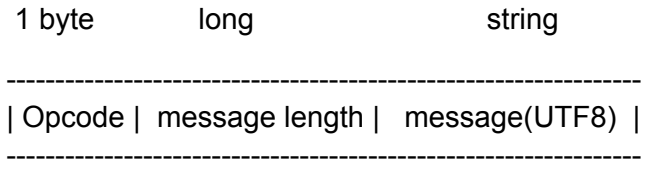


Figure 5-14: server error packet

- The message length is a long value in big endian representing the length of a message which describes the error encoded with UTF-8 charset.
- After this value, the packet contains a string representing error messages with UTF-8 encoding.

For example, when a client requests a connection to a server with a login which is already attributed, the server responds with error packet and message like “Login unavailable”.

**Opcode -3 :**

When a client A wants a private connection with client B, A send a request (cf: Figure 5-6). However, if the response to the request is negative (the receiver did not accept the connection) the opcode is -3 and it has the structure below :

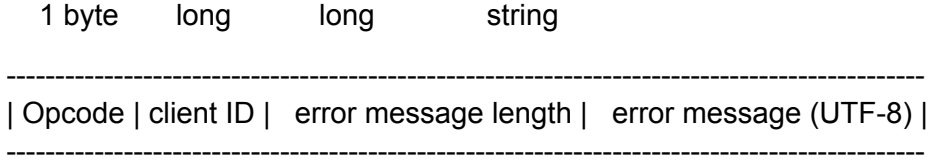


Figure 5-15: server negative response to request private connection packet

- The clientID is still a long and in big endian. It represents the ID of the client originally requesting the private connection.
- The error message length is a long value in big endian which represents how many bytes represent a string to describe why an error has occurred. If value of length is 0, it means an absence of description message
- If message length is greater than 0, it means the packet contains a string representing error messages encoded in UTF-8.

For packets with opcode -3, error message is optional, but message length is mandatory to know if a message is present or not.

**Opcode -4:**

When a client, for instance client A requests a private connection with client B, the server asks client B if he wants to accept an incoming connection. If client B refuses, he send the following packet containing opcode -4 to the server:

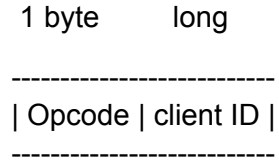


Figure 5-16: packet send to the server by client for negative response to private connection

- The clientID is the client's identifier who responds negatively to the server. It is a long in big endian as well.

**6. Normal termination**

The end of a connection between a client and the server or between two clients is marked by a disconnection request sent by a client who want to be disconnected.
To see the structure of this packet, please see figure 5-13.
When the server or a client receives a disconnection request, he must send an ACK and after that he closes the connection.

**Author's email address**

Christelle NGUYEN

Judicaël KOFFI

Email:

- christelle.nguyen97@gmail.com
- judkoffi@gmail.com