Network Programming
Group 2
M1 IT

Loïc Jouannisson
Tom Redon
UGE
February 2021

RFC of the ChatOS Protocol

Summary :

ChatOS is a very simple protocol used to transfer messages between users. It will be able to connect separately a group of persons in a general chat or just between 2 users in a private connection. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions that we took.

Acknowledgements :

The protocol was originally designed by Loïc Jouannisson and Tom Redon. It will evolve with the time and the suggestion given by the teaching staff.

The first version has been written between February and March 2021.

1 – Purpose :

ChatOS is a protocol that will authorize the users to connect to a IRC-type general chat, where every user will be able to see the messages posted in the chat from the other people, without saves of the chat between each connection.

It will also authorize people to create a private connection between them, without sharing their IP, either to chat between them or to use another protocol writing on top of TCP.

The protocol will be implemented on top of the Transmission Control Protocol, so it will be developed to be easy to use and will be reliable and will ensures an integrity to the users of the chatOS protocol.

2 – Overview of the protocol :

To start, the user will need to authentify, by sending is username to the server.

After that, he will be able to participate on the chat and will be authorize to send packets that will contains his messages.

He will receive an ACK packet that will confirm that the server received the packet with his message.

If the packet is not received or if there is an error in it, he will receive an error packet that will be able to determine the type of error that has been found in the packet.

If the client wants to create a private connection with another user, he will need to send a private connection ask request, the server will redirect it to the user.

The other user will respond with a private connection acceptation packet.

The server will contain some users’ data during their connection but will delete them when the collection will be ended.

3 – Relation to other protocols :

For now, the only relation with another protocol will be that the chatOS is implemented on top of the Transmission Control Protocol.

The packets will be developed to be usable on top of this protocol, and will have a header determining the length of the String when it’s needed, to be sure that we have a control on the receive of a packet.

4 – Initial Connection Protocol :

OPCode : 0 (Short)

Nickname : String ASCII

End of the String : 1 byte at 0

Firstly, the client will be obligated to connect on the server. He will need to send a packet with the upper information.

Firstly, there will be an OPCode (defined as 0) that will determines the type of the packet. The OPCode will be a Short data, to take only 2 bytes.

After that, the packet will contain the nickname of the user and will end the string with a byte written as a 0.

If the packet is viable, the user will receive an ACK packet and will be able to use the chat.

5 – Chat packet :

OPCode : 1 (Short)

ID of the packet : Long

Nickname or Channel : String ASCII

End of the String : 1 byte at 0

Message : String ASCII

End of the String : 1 byte at 0

To send a message in the chat or to another user, the client will need to send a packet formatted as follow.

There will be an OPCode (defined as 1) and this OPCode will be followed with the ID of the packet (number of packet sends by the user, will determine if the packet is well arrived and if not, if the client needs to resend it).

After that, the user will need to insert the String containing the nickname of the other user or a void string if he wants it in the general chat, ended by a byte to 0.

To finish, he will need to put his message, ended by a byte to 0.

6 – ACK Packet :

OPCode : 2 (Short)

ID of the packet : Long

The ACK packet is really short, but it’s intended.

We wanted a short packet, containing only the OPCode 2 and the ID of the packet to check if the packet was well received by the server.

7 – Error Code and Packet :

OPCode : 3 (Short)

Error Code : Short

Error Message : String

End of the String : 1 byte at 0

The error packet will contain an OPCode 3, an error code (detailed just under), an error message that will be ended by a byte to 0.

For the error codes, we will define them during the development but we can define some for now :

1 – Packet is missing data

2 – Packet has error in data

3 – The user asked doesn’t exist

4 – The user has not accepted a private connection with you

8 – Private asks packet :

OPCode : 4 (Short)

Nickname : String ASCII

End of the String : 1 byte at 0

The private asks packet will contain an OPCode 4.

If that’s the client that sends it to the server, it will contain the nickname of the user that the client wants to connect with, ended by a byte to 0.

On the other side, it will contain the nickname of the client that asks for the connection, ended by a byte to 0.

9 – Private connection acceptation packet :

OPCode : 5 (Short)

Yes/No : Boolean

Nickname : String ASCII

End of the String : 1 byte at 0

The packet will contain an OPCode at 5, a Boolean that will define if the user answers yes or no to the private connection.

It will also contain the nickname of the client that first asks for the private connection, and it will end with a byte to 0.

10 – Client Data :

The server will contain some data linked to the socket channel of the client that is connected.

In these data, it will register the nickname of the user, the last timing where the server received or send a packet to the user and the connections that have the user with the other clients.