Gustave Eiffel University
I. Madet
Request For Comments: 9207
N. Van Heusden
Category: Informational
March 2022

THE CFP PROTOCOL

Status of this Memo

This RFC specifies the implementation of the Chat Fusion Protocol, using the TCP protocol for the communication between users. It also defines the standardization of Chat Fusion implementation, states and statuses. Distribution of this memo is unlimited.

Summary

Chat Fusion is a chat service between several clients using the same server. The particularity of Chat Fusion is that it allows server to merge together in order to let users communicate between servers.

1. Purpose

The goal of Chat Fusion protocol is to allow multiple communications between client on same and different servers. Each client can choose to authenticate with or without a password. Once authenticated, clients can send public messages or direct message to another user. If a client sends a direct message to another user, they can attach a file to be transferred along the original message. Since server are able to merge together, we need to define rules to ensure there is no ambiguity with similar nicknames across servers:

- Clients authenticated on the same server must have different nicknames.

- Nicknames are discriminated with the server they belong to.

2. Overview of the Protocol

For every packet, strings are encoded using the UTF-8 charset. Files transferred are sent in byte chunks.

The CFP protocol is based on the TCP protocol, thus, a CFP connection is in fact a TCP connection between two machines. Moreover, this protocol relies on the reliability of TCP. Packets should never get lost.

Please note that every number are encoded in a big-endian system.

Every CFP packet starts with a code indicating the type of packet. For details about which code represents which packet, please see '(9) Packets Codes' section.

Each server is named on an integer (4 bytes) and cannot rename itself once started.

Most of the errors that could occur will result in an error response. However, if a packet received from a client is somehow malformed of corrupted in any way, the server will simply ignore this packet. This ensures a client is not able to make the server crash by sending bad packet to it.

3. Users Authentication

If a client wants to communicate with other clients over the Fusion Chat network, they need to authenticate either as a guest, only giving a username (as long as this username is not already taken by another client connected to the server), or as a registered user by giving a nickname and password combination.

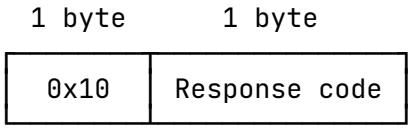
A client authenticating as a guest must send an 'AUTHENTICATION_GUEST' packet:

1 byte	4 bytes	n bytes
0x00	Nickname size	Nickname

Whereas, a client authenticating as a registered user must send a 'AUTHENTICATION_USER' packet:

1 byte	4 bytes	n bytes	4 bytes	n bytes
0x01	Nickname size	Nickname	Password size	Password

Once the server has received the authentication packet, it will process credentials and reply an 'AUTHENTICATION_RESPONSE' packet to the client to let it know if the authentication succeeded or not:



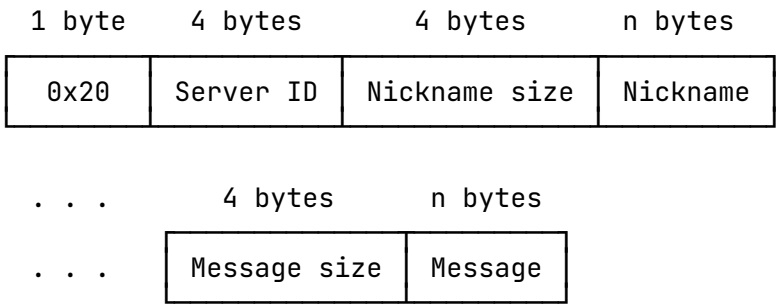
The response code can be one of the followings:

- 0x00 : Authentication succeeded.
- 0x01 : Authentication failed due to nickname already been taken. In other words, a registered user chose this nickname.
- 0x02 : Authentication failed due to incorrect nickname and password combination.
- 0x03 : Authentication failed due to the nickname being use at this very moment by another client.

If everything went well, at this point the client and the server has established a connection. The client is now able to send messages over the Fusion Chat network.

4. Public Messages

If a client wants to send a message to everyone over the Fusion Chat network, they need to send a 'PUBLIC_MESSAGE' packet:

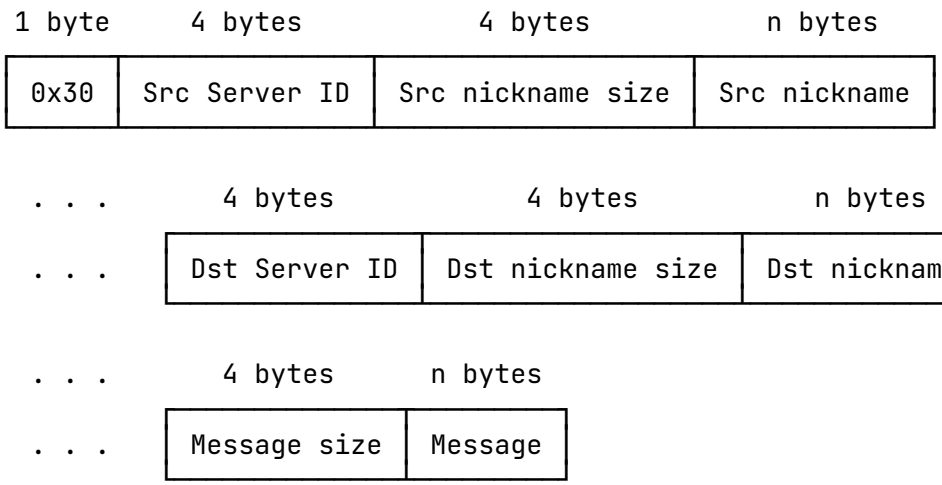


Just as a reminder, message must be encoded using the UTF-8 charset, therefore, the message size is the size of the encoded message.

Once the server has received the packet, it will dispatch it to every another client connected to the server, and throughout the network by forwarding it to its leader or to all its neighbors.

5. Private Messages

A client can also directly send a message to another client by giving its identity so that the server knows where to send the packet. Since the protocol allows server merges, the client must give the destination server in addition of the client identity. To do so, the client must send a 'PRIVATE_MESSAGE' packet:

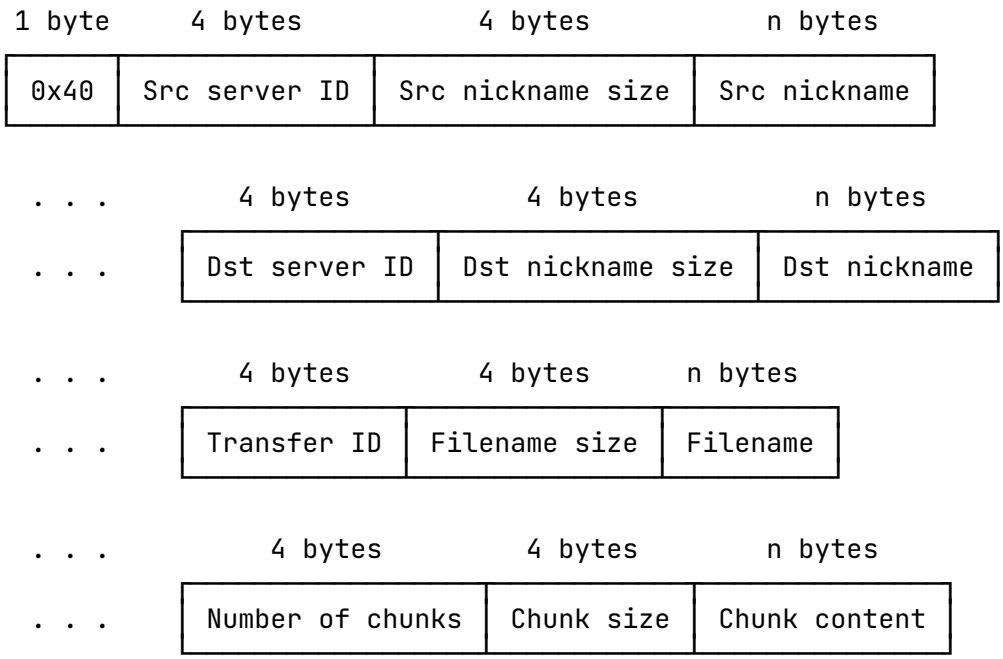


Once the server has received the packet, it will either send it to desired client if the server ID is the current server one, or forward it to its leader or to all its neighbors if it is leader.

6. File Transfers

In addition to sending direct messages, clients can transfer files between them. Files are sent in byte chunks of a maximal size of one kilobyte (1,024 bytes).

To do so, the client must send a 'FILE_CHUNK' packet. Please note that every transfer is identified by an integer, this ensures two transfers between the same client with the same filename and file size will not clash.



Once the server has received this packet, it acts just like message, forwarding it to the client or to the leader.

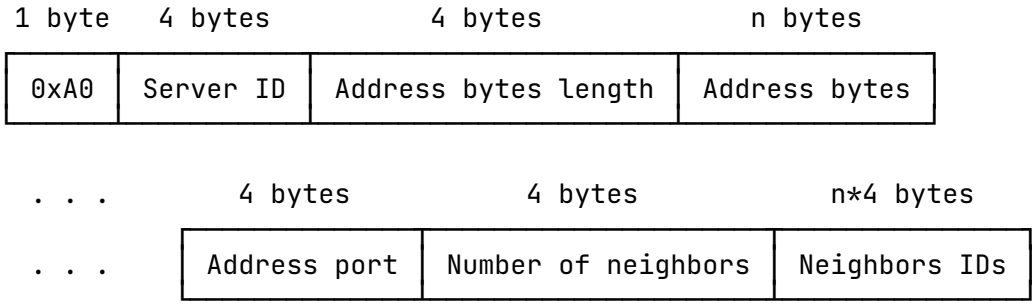
Client receiving these packets will rebuild the file using chunks content. The name is used to save the file using its original name (Extension included).

7. Server merge

The fusion system relies on a leader/slaves concept. When two or more servers are merged together, one of them is elected as the leader. The leader manages the fusion process and is responsible for packet traveling throughout the network.

Note: the leader is ALWAYS the server with the lowest ID.

If the server 1 want to merge with server 2, it must send a 'MERGE_INIT' packet to server 2. This packet contains the server 1's ID, its socket address and a list of all its neighbors IDs:



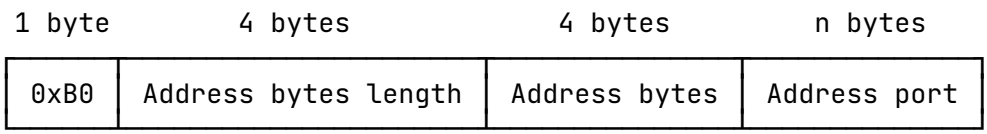
When sever 2 receives this packet, it will look for common neighbors and if none is found, it will respond with a 'FUSION_INIT_OK' packet. This last packet is basically the same as the 'MERGE_INIT' packet, with the code '0xA1'.

If a common neighbor is found, it will send a 'MERGE_INIT_KO' packet.

Now that two servers are merged, server 1 is elected as the leader. From now on, if server 2 wants to merge with another server OR if another server wants to merge with server 2, fusion process must be handled by the leader.

a. Non-leader trying to merge

If server 2 wants to merge with another server, it will send a 'MERGE_REQUEST' packet to its leader with the address of the server it wants to merge with:



The leader, receiving this packet, will attempt a fusion with the server. After the process, the leader will respond back with a 'MERGE_REQUEST_RESPONSE' containing a byte indicating if the fusion succeeded or not.

b. Server trying to merge with non-leader

If another server wants to merge with server 2, when receiving the 'FUSION_INIT' packet, the server 2 will respond with a 'FUSION_INIT_FORWARD' containing the address of its leader. The other server receiving this packet will try again with the new address.

c. Leader changing propagation

When a fusion occurs between two groups of servers, one of the two leaders will be demoted to a normal server. So, this server will send a 'FUSION_CHANGE_LEADER' to all its neighbors, telling them to connect to the new leader. This packet contains the new leader's address. Every server receiving this packet will try to connect to the new leader, and send a 'FUSION_ACKNOWLEDGE_LEADER' packet to it. The new leader receiving this will consider this server as a neighbor.

8. Packets Codes

This section summarizes all codes used in packets header. Client can send the following packets.

a. Authentication packets (0x00 & 0x10):

- 0x00 (AUTHENTICATION_GUEST) : Authentication as a guest
- 0x01 (AUTHENTICATION_USER) : Authentication as a registered user
- 0x10 (AUTHENTICATION_RESPONSE) : Authentication response

b. Public message (0x20):

- 0x20 (PUBLIC_MESSAGE) : Public message

c. Private message (0x30):

- 0x30 (PRIVATE_MESSAGE) : Direct message

d. File transfer (0x40):

- 0x40 (FILECHNK) : File chunk transfer

e. Fusion process (0xA0):

- 0xA0 (FUSION_INIT) : Initialization of fusion process
- 0xA1 (FUSION_INIT_OK) : Initialization of fusion process succeeded
- 0xA2 (FUSION_INIT_KO) : Initialization of fusion process failed
- 0xA3 (FUSION_INIT_FORWARD) : Forwarding of fusion init

f. Fusion request (0xB0):

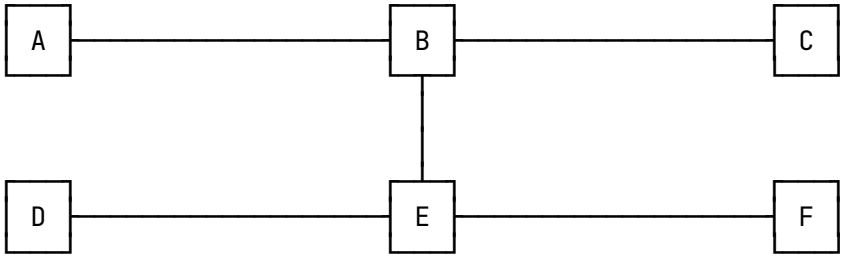
- 0xB0 (FUSION_REQUEST) : Request to merge
- 0xA1 (FUSION_REQUEST_RESPONSE) : Response for a fusion request

g. Leader changing (0xC0):

- 0xC0 (FUSION_CHANGE_LEADER) : Changing leader
- 0xC1 (FUSION_ACKNOWLEDGE_LEADER) : Acknowledgement of new leader

9. Annex

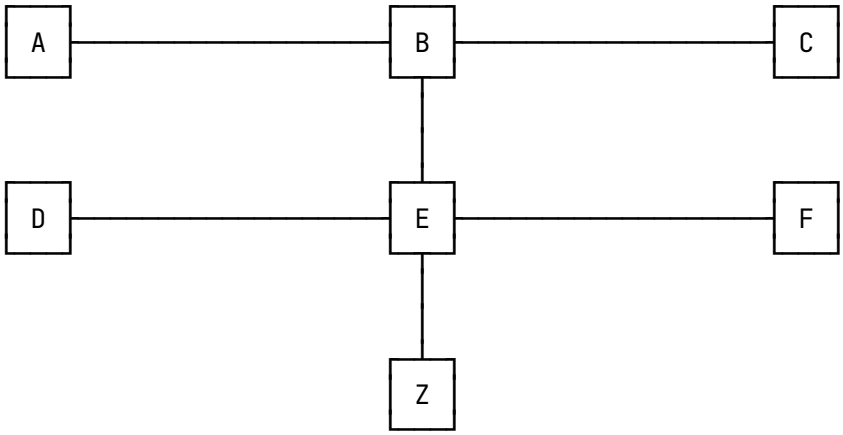
Talking about the merge system, we wanted to implement a smart routed mesh, using routing tables updating accordingly to new connections among the Chat Fusion network. The goal was to keep a map of server ID, mapped to the next server to follow and the remaining length of the path. For example, let six servers connected like shown:



Each server would have a routing table to know which server to follow to get to any server on the network. For example, let's see the routing table of server E:

	Next server	Path length
A	B	1
B	B	0
C	B	1
D	D	0
F	F	0

Now let's pretend a new server Z wants to merge with E. The two servers would exchange their routing table to update the shortest paths. The network would look like this:



The Z server would receive the routing table of E, and for each server it doesn't have a path to get to, it will save that E is the next server to follow. The Z routing table would look like this:

	Next server	Path length
A	E	2
B	E	1
C	E	2
D	E	1
E	E	0
F	E	1

On its side, the server E would add Z in its routing table and send this information to all its sibling in order to let them update their routing table, and so on recursively.

This system, although complex, is pretty efficient in terms of concurrent connections and packet traveling effectively. It should also guarantee that packets would never loop through the network, always following the right path. However, every new connection or lost connection has to be broadcast all over the network in order for every server to acknowledge the new network state. This complexity mixed with the small scale expected of a Chat Fusion network led to a simpler, yet working dense mesh.

11. Authors

Irwin Madet • email: irwin.madet@edu.univ-eiffel.fr

Nicolas Van Heusden • email: nicolas.van-heusden@edu.univ-eiffel.fr