Gustave Eiffel University
Request For Comments: 9207
Category: Informational

I. Madet
N. Van Heusden
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 'AUTHGST' packet:

1 byte	4 bytes	n bytes
0x00	Nickname size	Nickname

Whereas, a client authenticating as a registered user must send a 'AUTHUSR' 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 'AUTHRSP' packet to the client to let it knows if the authentication succeeded or not:

1 byte	1 byte
0x10	Response code

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 'MSGPBL' packet:

1 byte	4 bytes	n bytes
0x20	Message size	Message

Just as a reminder, message must be encoded using the UTF-8 charset. Therefore, the message size is the size of the encoded message. Note that the client DOES NOT give its identity in the packet, since the server has already authenticated it.

Once the server has received the packet, it will send a 'MSGPBLINC' to all its connected clients, consisting of the sender's identity followed by the message. The packet also contains the origin server id to allow any client to tell from which server the message was sent:

1 byte	4 bytes	4 bytes	n bytes	. . .
0x30	Server ID	Nickname size	Nickname	. . .

. . .	4 bytes	n bytes
. . .	Message size	Message

Clients receiving this packet will process it to display the name of the sender and the message in their console.

In another hand, the server will send a 'PCKFWD' to every server it is connected to over the Chat Fusion network. This is due to the merge system of the Chat Fusion protocol. This 'merge' system will be discussed further in this document. The packet is basically the original packet of size n, prefixed with the 'PCKFWD' code and the destination server ID:

1 byte	4 bytes	n bytes
0x50	Server ID	MSGINC content

This packet also contains the destination server ID to allow an evolution of the Chat Fusion network that could work on a routing system. Please see '(10) Annex' section to get more details about this evolution.

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 'MSGPRV' packet:

1 byte	4 bytes	4 bytes	n bytes	. . .
0x21	Server ID	Nickname size	Nickname	. . .

. . .	4 bytes	n bytes
. . .	Message size	Message

Please note that this packet is extremely similar to the 'MSGPBLINC' packet, however their content and purpose are totally different, resulting in a different packet code.

Once the server has received this packet, it will process it to determine the destination server. Then it will build a 'MSGPRVINC' packet consisting of the origin server ID, the sender nickname, the receiver nickname and the message:

1 byte	4 bytes	4 bytes	. . .
0x31	Server ID	Sender Nickname size	. . .

. . .	n bytes	4 bytes	n bytes
. . .	Sender Nickname	Message size	Message

Once the server has built this packet, it sends it to the correct server using a 'PCKFWD' packet:

1 byte	4 bytes	n bytes
0x50	Server ID	MSGPRVINC content

When the destination server has received the 'PCKFWD' packet, it will extract the 'MSGPRVINC' and send it to the correct client.

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 'FILECHNK' packet consisting of the origin server id, followed by the nickname of the receiver, followed by the file name, and finally the total number of chunks and its content. 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.

1 byte	4 bytes	4 bytes	n bytes	. . .
0x40	Server ID	Nickname size	Nickname	. . .

. . .	4 bytes	4 bytes	n bytes	. . .
. . .	Transfer ID	Filename size	Filename	. . .

. . .	4 bytes	1,024 bytes	. . .
. . .	Number of chunks	Chunk content	. . .

Once the server has received this packet, if the server ID differs, the server forwards this packet to the desired server through a 'PCKFWD' packet.

1 byte	4 bytes	n bytes
0x50	Server ID	FILECHNK content

Once the correct server has received this packet, it will build a 'FILECHNKINC'. This packet has the origin server's ID instead of the destination server and the sender nickname appended to it.

1 byte	4 bytes	4 bytes	. . .
0x41	Server ID	Sender Nickname size	. . .

. . .	n bytes	4 bytes	4 bytes	. . .
. . .	Sender Nickname	Transfer ID	Filename size	. . .

. . .	n bytes	4 bytes	1,024 bytes	. . .
. . .	Filename	Number of chunks	Chunk content	. . .

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 merge system relies on a dense mesh. Indeed, every server is connected to every other server on the Chat Fusion network. Since this approach is easy to implement but not efficient on big structures, the total amount of connected servers in the same Chat Fusion network is limited to 10 servers. Please see '(10) annex' section for further details about these implementation choices.

If a server A wants to merge its network with another server B network, A needs to connect to B, just like a normal client, and send a 'MRGREQ' packet consisting of A's server ID and every connected siblings:

1 byte	4 bytes	4 bytes	n * 12 bytes
0xA0	Server ID	Number of siblings	Sibling description...

A sibling description is the sibling Server ID, its IPv4 address and its working port:

4 bytes	4 bytes	2 bytes
Sibling Server ID	Sibling IP Address	Sibling Port

If B accepts the merge, it will respond with the 'MRGRSP' packet with its own siblings:

1 byte	4 bytes	4 bytes	n * 12 bytes
0xA1	Server ID	Number of siblings	Sibling description...

Otherwise, if the two servers are already merged, B will respond with a 'MRGDECL' packet:

1 byte	1 byte
0xA2	Error code

Error codes are listed below:

- 0x00 : The servers are already merged.
- 0x01 : The merge would result in a server limit exceeding.
- 0x02 : One or multiples server names conflicts among the hypothetical merged network.

8. Global Errors

If an error occurs on data transfer, an 'ERROR' packet is sent back to the client.

1 byte	4 bytes	n bytes	1 byte
0xF0	Sender Nickname size	Sender Nickname	Error code

Error codes are listed below:

- 0x00 : Destination server is unknown or not connected.
- 0x01 : The receiver client is unknown or not connected.

9. Packets Codes

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

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

- 0x00 (AUTHGST) : Authentication as a guest.

- 0x01 (AUTHUSR) : Authentication as a registered user.

- 0x10 (AUTHRSP) : Authentication response.

b. Message sending (0x20):

- 0x20 (MSGPBL) : Global message (outgoing)

- 0x21 (MSGPRV) : Direct message (outgoing)

c. Message distribution (0x30):

- 0x30 (MSGPBLINC) : Public message distribution (incoming)

- 0x31 (MSGPRVINC) : Private message distribution (incoming)

d. File transfer (0x40):

- 0x40 (FILECHNK) : File chunk transfer (outgoing)

- 0x41 (FILECHNKINC) : File chunk transfer (incoming)

e. Packet distribution (0x50):

- 0x50 (PCKFWD) : Packet forwarding to another server.

f. Server fusion (0xA0):

- 0xA0 (MRGREQ) : Request to merge containing siblings’ info.

- 0xA1 (MRGRSP) : Merge request response containing siblings.

- 0xA2 (MRGDCL) : Merge request rejection.

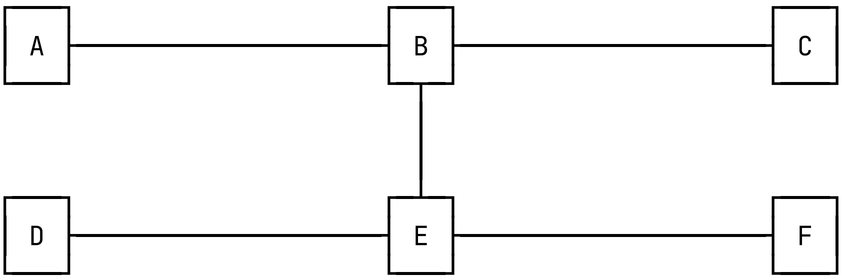
g. Errors (0xF0):

- 0xF0 (ERROR) : Packet containing error code.

Every packet marked as 'outgoing' is traveling FROM a client TO a server, whereas an 'incoming' packet is traveling FROM a server TO a client.

10. Annex

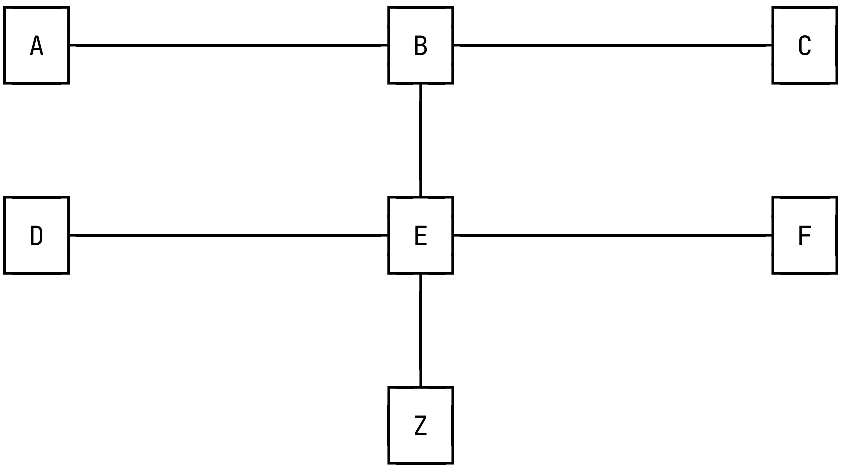
Talking about the merge system, we wanted to implement a smart routed mesh, using routing tables updating accordingly to new connections among the Chat Fusion network. The goal was to keep a map of server ID, mapped to the next server to follow and the remaining length of the path. For example, let six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