Distributed Systems and Computing : Laboration 2

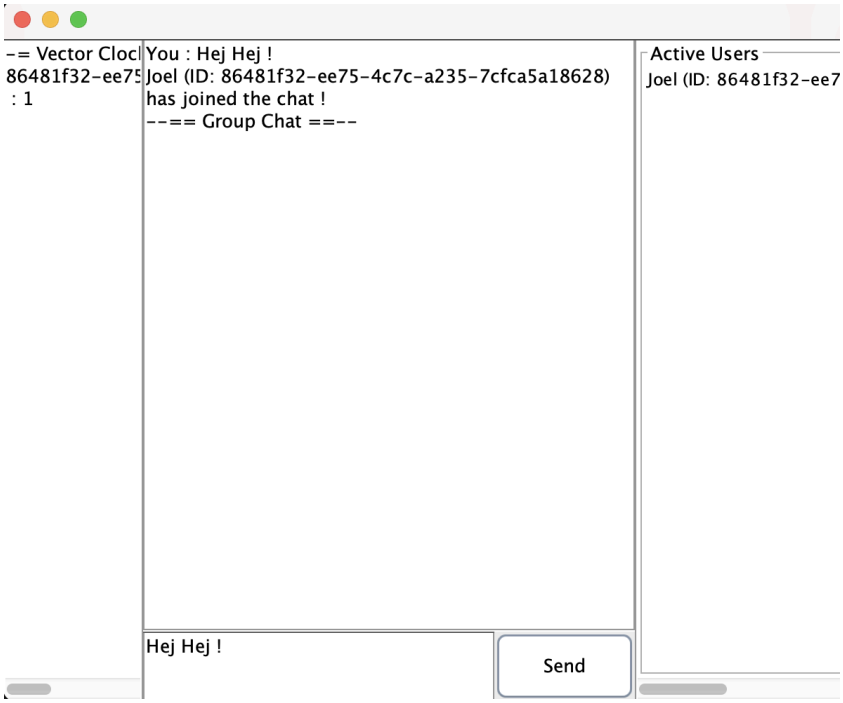
Supervisor : Nayeb Maleki

Mittuniversitetet
MID SWEDEN UNIVERSITY

MBIAPA KETCHECKMEN
Joël Trésor
Autumn Semester 2024

In this second laboration, we want to implement causal ordering in our chat system based on the implementation of the first laboration. The aim is to make each client individually keep track of the clock of each other client using a vector style clock. Therefore, in order to achieve this, we proceed step by step.

Firstly, we need to modify our previous user interface in order to be able to show the evolution of the vector clock and its correct behavior. This is the reason why we add a new area on the right side of the window in order to display the vector clock list of the client in the following format : `userID : localClockValue`



After this modification of the user interface, the next step is to implement the vector clock in order to implement the causal ordering. For that, we implement a class VectorClock that contains a map of string and integer corresponding to the vector clock, and additional functions that will be useful in our implementation : *constructors*, *getters*, *increment* and *update* of the vector clock as well as a *isCausallyOrde*r function to check later if the message receive can be display now or later. Once this is done, we add a vector clock attribute to each client in order to keep track of the activity in the group chat, a pending messages list to store the messages that cannot be delivered upon receiving a message, and a chat messages history list containing all the messages sent and received by the client. This list will store each message by their message ID (which is *senderID/number of the message based on the clock of the sender*) and the chat message.

When a client sends a message, it should send the chat message and a snapshot of the vector clock of the sender. Therefore, we modify the chat message class to include the vector clock. Additionally, when sending a message, we first need to increment the vector clock of the sender

before taking a snapshot of its vector clock and then send the chat message. When a chat message is received, we have to follow different steps :

- to avoid having duplicate messages we check if the message is in the chat messages history of the receiver based on the message ID. If not we can add it to the history and proceed the message, otherwise we just ignore it
- for each client receiving the message (except the sender), we check if the message is causally ordered based on the vector clock of the message and the local clock of the user
- if the message is causally ordered we can display it and update the local vector clock and the UI. Moreover, we also check if we can deliver some of the pending messages
- if the message is not causally ordered, we can't display it but put it on the waiting list. In this situation, we have to request for missing messages.

After noticing that a client misses messages from a specific user, he has to send a request for resending missing messages in order to recover the lost messages. Using the vector clock of the received message, the client can send a resend message request of the i missing messages to the others clients. The client also uses this opportunity to check if he also misses messages from other clients and not only the sender.

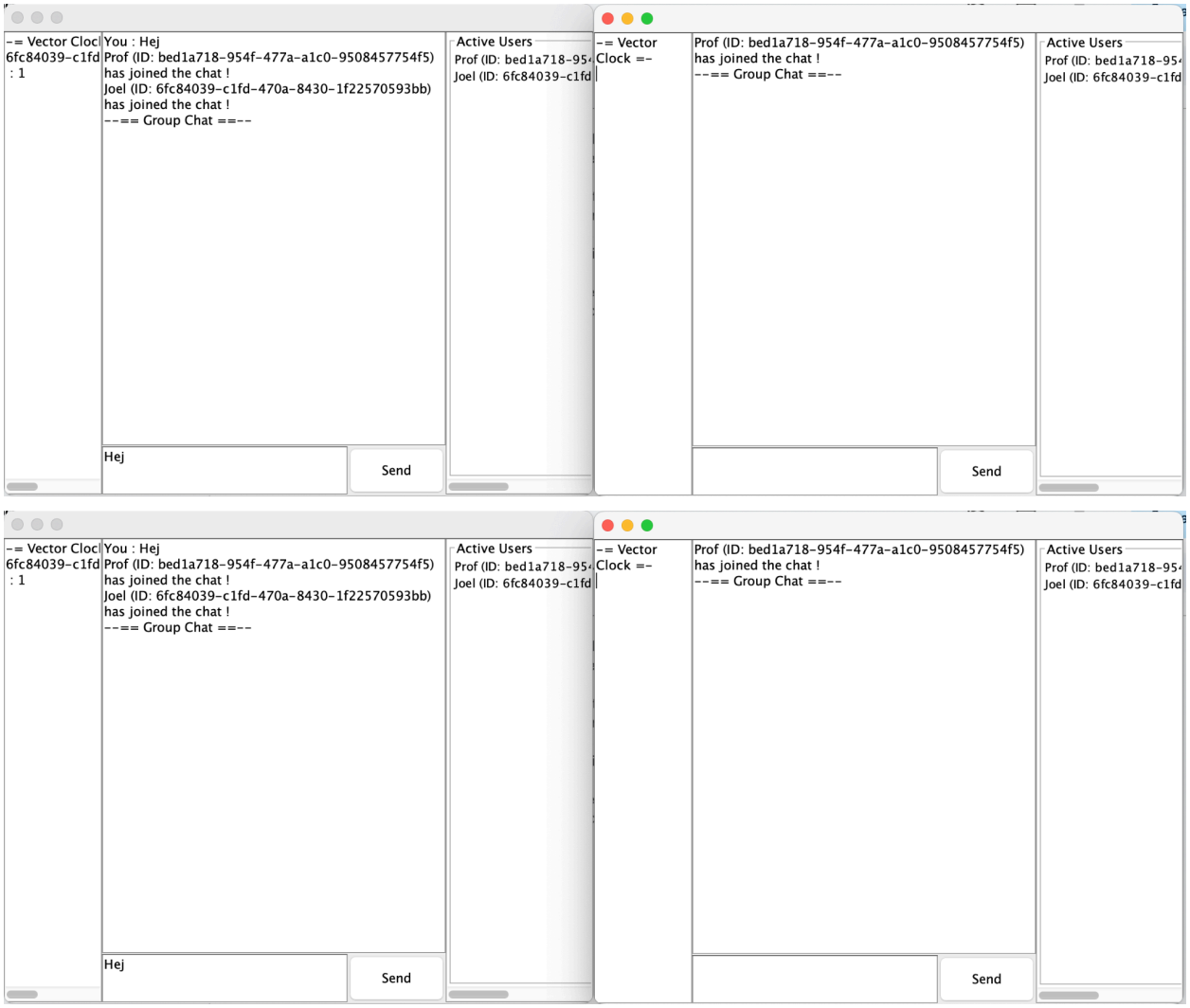
When a resend request message is received, the receiver of the request will :

- build the id of the chat messages lost based on the information received
- check if this specific message is present in its own messages history
- recover the message and send a resend response message to other clients

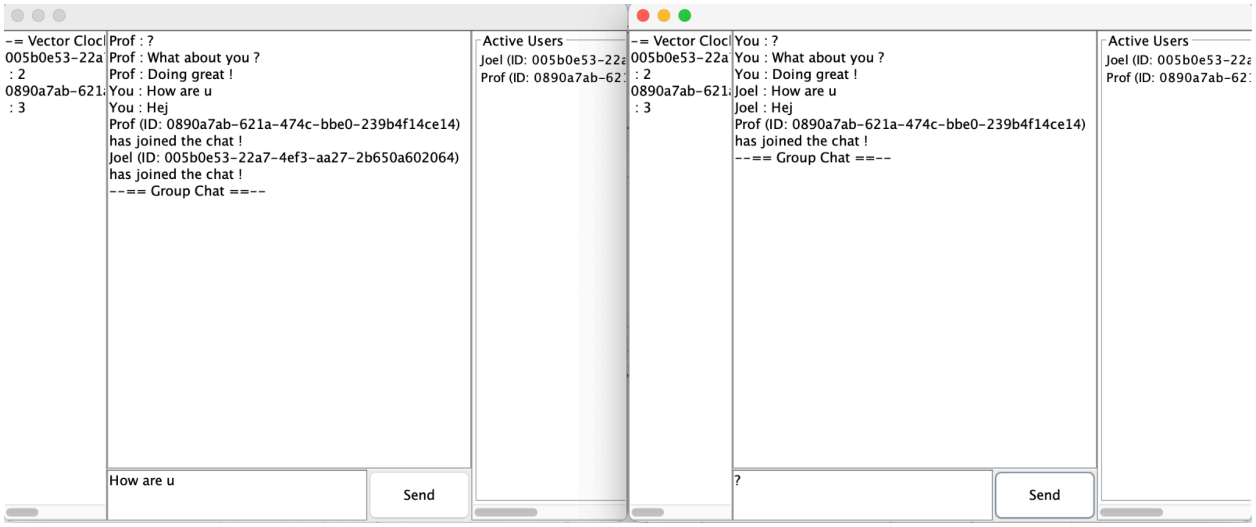
The resend response message cannot be lost, we can only drop packages regarding the chat message. Therefore, we proceed the resend response message exactly like a chat message : if the message is not in the history list, we check if it is causally ordered, display it and check if pending messages can be displayed. Otherwise we put it in the pending list.

This will ensure that all the lost messages are requested by a client, resend by others clients, and displayed on the screen in the correct order without having duplicated messages.

Here is an example of visualization step by step with 50% of chance to drop the packets. We first show a situation where there are only two users, and then a third user joins the sessions. User Joel sends a first message which is lost, and then sends a second message which is not lost. Because the message is received and not causally ordered by the user Prof (right), he sends a request to recover the missing message before displaying the message received.



The next two messages sent by the user Prof are also lost, but not its third one. Following the process, user Joel will request the missing messages before printing the last received :



A third user, John Doe joins the session, and the user Joel sends a message which is not lost by user John Doe. Because the message received isn't causally ordered, the user John Doe will send a request for missing messages. At each step, we can see the evolution of the vector clock for each user in each client window.