Design Document

The algorithm we used for the sending and receiving of reliable messages was Reliable Multicast. Essentially, every process (user) is a server and a client. When a user wants to send a message, it basic multicasts it and when a process receives it, if it hasn't seen it before, it will basic multicast the message to everyone or else it does nothing. This way either all correct processes get the message or none of them do. Alternatives to this is Basic Multicast or Unicast but neither of these are reliable so they are not sufficient for our problem statement.

We implemented the ISIS algorithm for total ordering. When a user wants to send a message, it first multicasts a declaration that it wants to send a message. Then all of the processes, send back at what time they would like the message to be delivered. The sender process then takes the latest time out of the values and multicasts a message to all of the processes at that time. On receiving this message, every process inserts the message into its local priority queue ordered by time and if the time is right, we deliver the message. An alternative would have be the Sequencer method. We did not choose to implement this because it has a central point of failure. We could have modified it to not have a single point of failure, but at that point, the implementation seemed as complicated as ISIS and brought no benefits.

We used All-to-All heartbeat failure detection. Essentially every T seconds, we send a heartbeat to all of the other processes letting them know we are there. If a process hasn't sent a heartbeat for a while, the process is marked as dead and the process that discovers this notifies everyone. Our worst case detection time is 2 seconds. We make our worst case detection time lenient to help ensure that there are no false alarms and because notifying the user immediately is not necessary. An alternative would have been to notify others of a failure and have the other processes send confirmation messages to confirm failure or send a message to claim that the process is in fact alive. We chose the first option because we did not want our failure detection to be expensive since it is not the main part of the application. We did not want failure detection to take up a lot of bandwidth. We chose All-to-All to ensure completeness because we needed to be able to handle multiple simultaneous failures so ring-based and centralized failure detectors were not feasible.

The amount of messages sent for any reliable multicast is N^2 Unicast messages where N is the number of users because all N processes could potentially send N messages.

The amount of messages sent every T seconds for failure detection is N^2 messages where N is the number of users in the group chat. This is because every T time units, every process sends N heartbeat messages to the N processes.

The amount of messages sent to deliver one message using the ISIS algorithm is the following:

N messages for a process to declare it wants to send a message, N messages to send a proposed priority to the original sender, and N^2 messages to ensure that the agreed priority is told to everyone. This leads to a total of $N^2 + 2N$ messages.

We tested our design by sending messages on different VMs and ensuring all correct processes receive a message and it is the same order for every process. We also tested the code against network delays to make sure we still had similar behavior.

Our code was not able to handle network delays so we were not able to conduct solid testing as network delays increase.