Implement a distributed group chat application. Users can create and delete rooms. For each room, the set of participants is specified at creation time and is never modified. Users can post new messages for a room they are participating to. Within each room, messages should be delivered in causal order.

The application should be fully distributed, meaning that user clients should exchange messages without relying on any centralized server.

The application should be highly available, meaning that users should be able to use the chat (read and write messages) even if they are temporarily disconnected from the network.

The project can be implemented as a real distributed application (for example, in Java) or it can be simulated using OmNet++.

Assumptions

- Clients are reliable but they can join and leave the network at any time. Network failures and partitions may happen.

Rules

1. The project is optional and, if correctly developed, contributes by increasing the final score.
2. Projects must be developed in groups composed of a minimum of two and a maximum of three students.
3. The set of projects described below are valid for this academic year only. This means that they have to be presented before the last official exam session of this academic year.
4. Students are expected to demonstrate their projects using their own notebooks (at least two) connected in a LAN (wired or wireless) to show that everything works in a really distributed scenario.
5. To present their work, students are expected to use a few slides describing the software and run-time architecture of their solution.
6. Projects developed in Java cannot use networking technologies other than sockets (TCP or UDP, unicast or multicast) or RMI.
7. Students interested in doing their thesis in the area of distributed systems should contact Prof. Cugola for research projects that will substitute the course project.

Highly available, causally ordered group chat

Project requirements

- Distributed chat application
- A user can create a room specifying a non mutable participants list
- The user who created the room can delete it
- Every user inside a room can post messages
- Messages should be delivered in causal order
- Users can use the chat even if they are temporarily disconnected

Implementation

To simulate this scenario we used Omnet++

To achieve the requirement of messages causality we implemented a protocol based on vector clocks.

High availability is guaranteed by allowing each client to ask for missing messages to every client being part of the room.

Handling missing packets

In case a client (due to a network partition, low time to live, ecc…) doesn't get a message he needs he can request it to other clients with an AskMessages packet.

Every client in possession of that message can then send that message to the requester.

In case a room gets created and the room creation message gets lost the admin of the room will continue to send the message until eventually the client will answer with an ack message.

In case of room deletion message miss the client can receive it from any client when asking for messages.

Network partitions

In our simulation a network partition can occur.

We cut communication for a random set of gates and after t time the communication gets restored.

Testing

In order to test that our simulations achieves its requirements we had to test for consistency.

We used a complementary Python 3 script that parses simulation output and checks for:

- Causality (If a message gets displayed by a client, such client must also have already displayed every message the sender has seen before sending that message)
- Every message gets delivered to every client in the room and only to those clients.

Testing (scenarios)

We created three different network topologies to test that our clients can work under different network circumstances.

- Ring with 5, 10 and 50 clients
- Fully Connected with 5, 10 and 50 clients
- Tree with 5, 10 and 50 clients

With events timing specified in Network.ned and omnetpp.ini.

Each simulation with more repetitions.

Testing (scenarios) - events for repetitions 0

	num partitions	num rooms	messages	max messages in a room
Ring - 5	42	44	1200	29
Ring - 10	45	76	5489	80
Ring - 50	2	29	4356	20
Fully - 5	42	51	1233	46
Fully 10	45	80	5427	38
Fully - 50	3	26	2658	26
Tree - 5	41	41	1250	88
Tree - 10	37	86	4942	39
Tree - 50	2	23	5685	40

Demo