Exercise 5

Multicast Chat

1 Goal

The goal of this exercise is to implement a chat application using the low-level network APIs of Java. In this application, each client must be able to (1) create a chat room, (2) join/leave a chat room, (3) list the participants of the chat room it is interested in and (4) send messages to all the clients of its chat room. A chat client can be interested in one chat room at a time.

2 Architecture

We depict in Figure 1 the architecture that your chat application must follow.

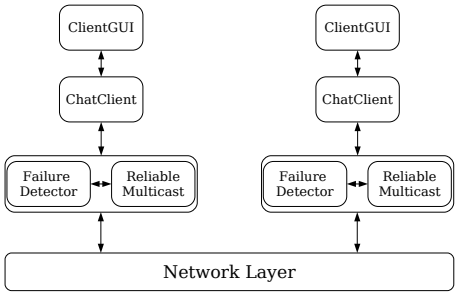

Figure 1: Multicast Chat Architecture

Your chat application must consist of: (1) a graphical user interface (`ChatGui` see Figure 2), (2) a controller `ChatClient` as well as (3) a reliable multicast layer. This layer can be decomposed in two main parts: a failure detector and a reliable multicast.

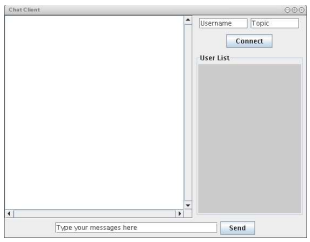

Figure 2: A Chat Client GUI

Note that your implementation must allow to change the implementation of the failure detector as well as the reliable multicast seamlessly (i.e., with minimum changes in your code).

3 Specification

We present here the informal specification that your implementation must fulfill.

A chat client must be able to subscribe to a chat room corresponding to its interest. A chat client is successfully subscribed to a chat room as soon as it is ready to receive messages of the topic of the chat room. Once subscribed, a chat client should get the list of the other subscribers.

When a chat client wants to send a message, this message is sent to all the correct subscribed chat clients available in the list.

The detection of correct chat clients depends on the output of an unreliable (best-effort) failure detector (i.e., after some time, when a chat client p_i as no information about another chat client p_j in its list, p_i decides that p_j has crashed).

The chat application is peer-to-peer (P2P), i.e., no central server must be relied upon.

4 Guidelines

Your chat application must use the IP multicast layer. We assume that the size of the messages that are sent between the chat clients is less than 64KB. All messages are sent to a well known IP multicast address and port (given through the command line) and when receiving the messages the clients filter them according to their interests (i.e., corresponding to the chat room they subscribed to).

The implementation of your failure detector should make realistic timing assumptions about the communication delays. Otherwise, every chat client can obviously be detected as faulty even if it has not crashed.

5 Questions

Here are several questions you must answer by e-mail when sending back your implementation (we still assume a P2P network and the IP multicast unreliable properties):

1. Is it possible, according to the above specification, that two correct chat clients receive two distinct messages in a different order? Why?
2. How can you deal with messages' sizes greater than 64KB (i.e., if the messages have to be fragmented)?
3. What hypothesis should be done on the system if we want a message to be delivered to all the effectively correct chat clients (and not only to the clients detected as correct by the failure detector)?

6 Due

You have to give, for June 27^{th}, the complete source code of your application. You can send your code, **without** the compiled classes, in a **.zip** file archive (or **.tgz**) to the following email address: *Sebastien.Baehni@epfl.ch*. The archive **has** to contain a script file used for compiling your code (either on Windows or on Linux). Your email should arrive before 1pm and its subject has to be: *IDS EX5*. Also, please put in the content of your mail the first name and name of your partners.

Furthermore your e-mail must contain the answers to the above questions. We will take into account your answers during the assignation of your final grades.

The application has to run correctly (i.e, without bugs) for you to receive 0.33 bonus points. If the code is well designed, well written and generic, you will receive additionally 0.33 points. Finally, if the code is well documented (i.e., javadoc preferably) you will get the final 0.33 points of the bonus.

References

[1] Java API Documentation. *http://java.sun.com/j2se/1.5.0/docs/api/index.html*.