Distributed Systems
Practical Project – 2015-2016

Totally-Ordered Multicast

The practical project will leverage the techniques and know-how acquired during the lectures and the work studies. The goal is to have a totally-ordered multicast over a dynamic group of peers. The project will be carried out over 2 weeks, in three milestones:

- Week 1: A message layer over NIO, with a simple broadcast over a group of peers
- Week 2: A totally-ordered multicast over a dynamic group of peers, with a burst testing mode.
- Week3: A working multicast, validated via a burst mode, with performance numbers.

Each week, before the classroom, you are expected to deliver a cut of your work, with the requested functionalities for that milestone. If you have problems, we will discuss them together in the classroom, helping you overcome them and move forward.

However, be advised that your final grade will be the result of the evaluation of the three cuts, not just the last one. A steady effort over the course of the three weeks is required of you, not a last minute effort.

Each cut must be an Eclipse project. The project must be named by the last names of the group members (Gruber.DePalma par example). Please include a README with the following points:

- How to launch the project and run tests
- An quick overview of the role of the main Java classes used of your implementation

Your code must be commented and the comments must be useful.

Week 1: **A message layer over NIO, with a simple broadcast over a group of peers.**

You will surrender a first cut of your code before **Tuesday, January 5th, before midnight**.

The distributed architecture must be a peer-to-peer architecture, where all peers are identical, running the same code. In the first week, the group of peers will be given to each peer, as a list of IP addresses and ports.

You are given an abstract message layer, allowing to send and receive messages, designed following an event-oriented style. You must respect the given abstract framework for the message layer and use it to wrap the Java NIO library. This is a small and easy step, considering all the code you have been given and that you already read and understood before the holidays.

Testing: To be able to debug and test, you will write a simple broadcast over a group of at least 3 peers. Each peer will have a broadcast thread that will iteratively send messages to all peers in the group.

Week 2: A totally-ordered multicast over a dynamic group of peers, with a burst testing mode.

You will surrender the second cut of your work before **Tuesday, Sunday 10th, before midnight**.

The second cut is a totally-ordered multicast, using Lamport's logical clocks, as we have seen in our lectures and practical studied. The multicast will work over a dynamic group of peers. Each new peer will be given the IP and port of one peer of the group and will request to join the group, following the design we discussed in the classroom.

This is again a small coding step, but likely testing will be more challenging. You will use a self-tested burst mode. Each peer will use a burst thread to send messages as fast as possible, using messages of random sizes. In other words, peers will send messages concurrently, stressing the totally-ordered multicast and the message layers.

The burst mode must be self-tested. First, you must verify that received messages are correct, with the correct contents. One possible approach is to use a checksum. Second, you must verify that all messages are delivered to all peers and in the same order. One possible approach is to have one of the peer acting as the verifier. Outside of the totally-ordered multicast, all peers send to the verifier the order in which they deliver messages (sending the logical clock of delivered messages, along with the sender and receiver peers of these messages). Thus, the verifier can easily check that all peers are delivering all messages in the same order.

Week 3: A working multicast, validated via a burst mode, with performance numbers.

You will surrender the third and final cut of your code before **Tuesday, Sunday 17th, before midnight**.

In this final cut, everything is expected to work, both the totally-ordered multicast and the self-testing burst mode. You are asked to gather basic performance numbers: latency and bandwidth, in burst mode, for several message length (64, 256, 512, and 1024 bytes)

A fully implemented totally-ordered multicast with a self-tested burst mode is mandatory to expect a grade over 10/20. You do not have to write a specification document. You do not have to write a design document, but you must comment your code and include your design choices. If your implementation still has problems (bugs), you are asked to describe them, write up an analysis of the bugs, and a description of the potential fixes if you can envision them. The quality of your code, the comments, and the analysis of the remaining problems will be the determining factors of your final grade.

Non negotiable points:

- **<u>The implementation must be based on the Eclipse project provided to you</u>**. You may use NetBeans for your own development, but you are required to surrender an Eclipse-based project.
- The project must be self contained, based on Java 7. Everything should be included in the project, source code but also any documents.
- **<u>The name of the project must be composed of the last names of your group.</u>**

There must be a document (pdf) in the project explaining the following:

- The specification of your totally-ordered multicast
- The design of your fault-tolerant totally-ordered multicast
- How to launch the project and run tests
- Where are the main entry points (classes, methods) in the code
- The overall class design of your implementation
- Other main points that are important in order to understand your code

If your project does not work fully, you must be able to explain its design and explain why it does not. In other words, you must be able to summarize your design, being clear and concise about its main points, and analyze the origin of the problems remaining in your implementation.

You must work in groups of two or three students. Any student in a group must be able to explain all parts and all facets of the project; each project will be defended by the group, each students answering individually different questions on the project.