# Algoritmos e Sistemas Distribuídos- Project Phase 3

João Carlos Antunes Leitão

NOVA Laboratory for Computer Science and Informatics (NOVA LINCS)

and

Departamento de Informática

Faculdade de Ciências e Tecnologia

Universidade NOVA de Lisboa

V 1.0 $\alpha$

$22^{th}$ November 2019

## 1 Overview

This document discusses the third and final phase of the ASD project for 2019/20. The project has three phases which will iterate the system being designed, enriching some components to improve their performance, or adding new functionality. The overall goal of the project is to build a robust and efficient publish/subscribe system based on Topics. The third phase is explicitly over simplified due to time constraints and conflicts with other projects.
A publish/subscribe system is a distributed system that offers decoupling between message generators (or senders) and message consumers (or receivers). This decoupling is both achieved in terms of space (hence the distributed component) and time (as the receiver and the sender do not need to be simultaneously on-line for a message to be transmitted). In the first phase of the project we will be focusing on the decoupling on space (time will start to be tackled on the second phase). In topic-based publish/subscribe systems, each message that is *published* is tagged with one (or more) topics. Messages are delivered only to processes that have shown interest in received publications in one of the topics associated with the message through a process called *subscription*. A process that is no longer interested in received messages for a given topic, issues an *unsubscription* for that topic to inform the system of this. For simplicity, in this phase, we will consider that messages published in this system are tagged with exactly one topic. In this phase of the project, each message published across each topic will have a sequence number (associated with the topic) that is attributed by the coordinator of the topic (at the Publish/Subscribe layer). This implies that every message published in each topic has to be sent first to the coordinator of the topic and only after that can be disseminated. Publications should be delivered across nodes respecting that total order i.e., a subscriber for topic $T$ that delivers message $m$ with sequence number $7$ of topic $T$ cannot deliver message $m'$ with sequence number $9$ of topic $T$ before delivering the message with sequence number $8$ for that topic (there are no restrictions across different topics).
In the following the proposed protocol architecture (and their interfaces) for solving this phase of the project is presented (Section 2). The programming environment for developing the project is briefly discussed next (Section 3). Finally, the document concludes by providing some information on operation and delivery rules for this phase of the project (Section 4).

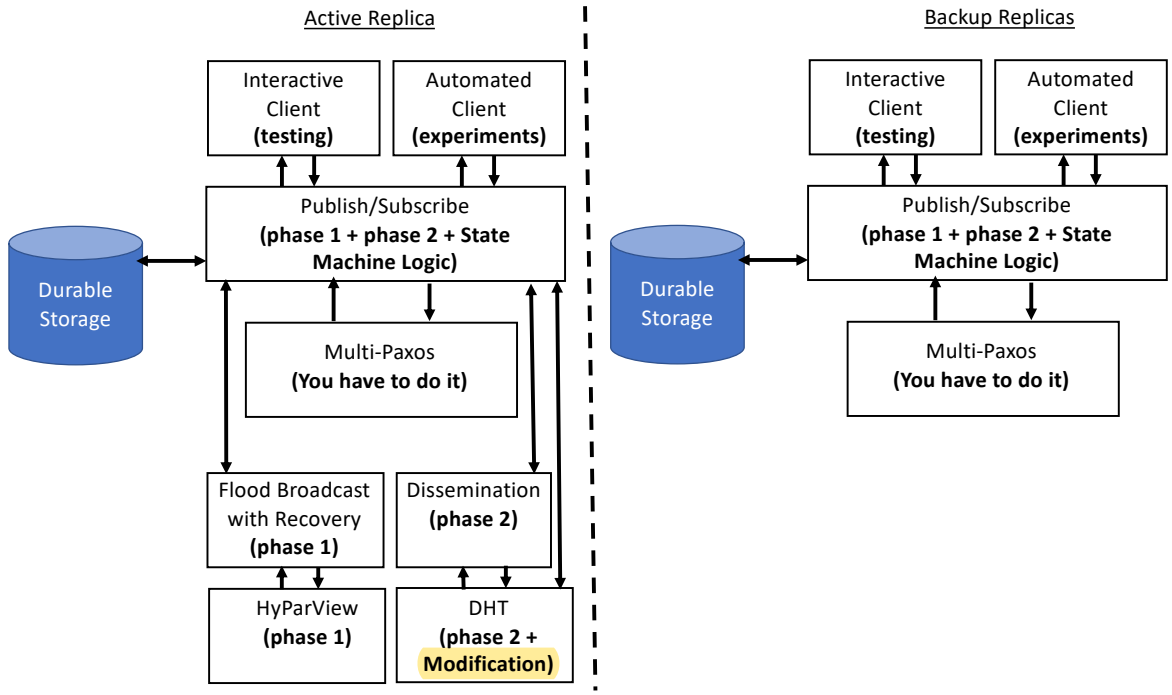


Figure 1: Proposed (abstract) composition of protocols to solve the project.

## 2 Solution Architecture

Figure 1 illustrates the proposed layering of protocols to solve the phase 3 of the project. The interactions between the protocols remain similar to the ones in the previous phases. The interface used by Multi-Paxos is the standard. Propose and Decided (tagged with the instance number). The goal of replication here is to have spare nodes (so automatic recovery is not handled in this project). Instead, for each node that actively belongs to the system (according to the specification of phase 2) the students are expected to be able to run two additional replicas of the Publish/Subscribe system, that replicate the state through multi-paxos. Note that in this architecture, the Publish/Subscribe protocol evolves to become the state-machine being replicated.
The State-Machine should support the addition of new replicas to the system, and should be able to remove replicas that fail. In the case of the failure of the replica that is part of the overlays (and hence, fully integrated in the system), students are not expected to reconfigure the system, and a human operator would have to manually promote one of the previous replicas to enable the overlay and communication protocols.
The Durable Storage component in the Publish/Subscribe state machine is optional. Its goal is to record all messages published on each topic, to avoid an excessive consumption of memory by processes during experimentation. The implementation suggestion is that you have a file for each topic being managed by the replica.
Figure 2 denotes a high level view of a possible configuration for the system that should be constructed during this phase of the project.

### *2.1 Changes to the Publish/Subscribe Protocol*

The Publish/Subscribe Protocol will have to be extended to model a state machine and interact with Multi-Paxos. This implies that this protocol will now have to maintain information about its replica set (should be 3 replicas in stable conditions). The protocol should also allow new client operations to assist in enforcing all guarantees aimed by the system and also to provide some load balancing. In particular, it should allow a subscriber to explicitly request a particular message (i.e., getMessage) for a topic (given the topic and sequence number of the message), it should also allow any other node in the system to request the current membership of its replica set. This allows nodes to publish or

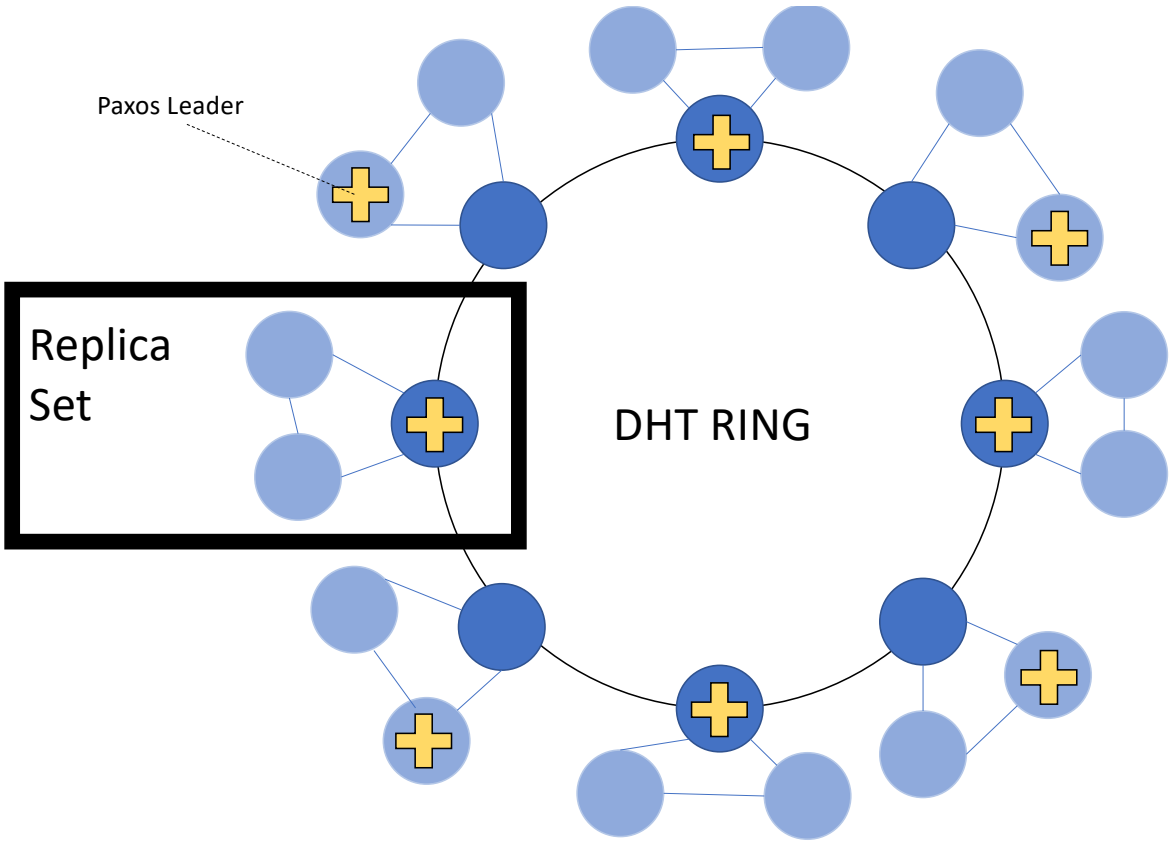


Figure 2: High Level view of the System Architecture.

issue getMessage operations to any replica in the replica set. Notice that both of these operations are read only (they do not modify the state of the state machine) and hence do not have to be ordered through paxos.
Whenever a replica (of the state-machine protocol) receives a request to publish a message, these have to be ordered through paxos to attribute a sequence number for the appropriate topic. This has to be done through paxos. Since we are using Multi-Paxos and hence have a distinguished proposer, every replica of the protocol must be aware of who is the current leader of Paxos, so that those requests can be redirected to the appropriate leader (the same is true for requests to add a new replica to the replica set).
The only optimization that students should consider if the want (**it is not mandatory**) is to batch multiple publish events into a single paxos instance. Notice that when one replica is added to the replica group, state transfer must be conducted. This can be easily achieved by sending a copy of the files in durable storage of one of the previously present replicas, and having the new replica parse those files for any information that it might need.

## *2.2 Details on the Multi-Paxos*

When implementing multi-paxos you should consider the following aspects:

There should be a special **Start Request** sent from the Publsh/Subscribe protocol to the Multi-Paxos protocol to indicate that the protocol should start operating and what is its initial state. Among other things, this request you should provide an optional membership parameter to Multi-Paxos. If no membership is provided, it means that that instance is the first replica in a replica group, and hence the size of the membership is one, meaning that the replica can make a prepare on its own and become leader immediately. If a membership is provided, then the local instance of Multi-Paxos was added later to the system and should perform state transfer from another replica. The membership parameter might indicate the current leader and sequence number, but this implies that

this membership information has to be provided by the Publish/Subscribe protocol after performing the addReplica operation for itself. Take advantage of the reply to that request to get all information necessary to bootstrap Multi-Paxos.

Multi-Paxos should also be able to process a request issued by the Publish/Subscribe protocol to indicate changes in the composition of the replica set (and the instance in paxos in which that change happened). That is important to allow Multi-Paxos to adjust the size of quorums used locally (remember that the majority of 2 is 2).

Multi-Paxos must notify the Publish/Subscribe layer whenever the leader changes. This is important so that the Publish/Subscribe protocol knows if it should redirect operations that modify its state to the leader or directly propose operations to the local instance of Multi-Paxos.

### *2.3 Overcoming Limitations in your previous implementations*

If you had issues with implementing components of the previous phase, particularly on the DHT layer, you can resort to a one-hop DHT /equivalent to a global membership). Similarly, you can rely on a point-to-point dissemination scheme if you were not able to implement a distributed dissemination solution operating over the DHT (similar to Scribe).

## 3 Programming Environment

The students will develop their project using the Java language (1.8 minimum). Development will be conducted using a framework developed in the context of the NOVA LINCS laboratory[1] written by João Leitão, Pedro Fouto, and Pedro Ákos Costa, whose internal code-name is **Babel**.
The framework resorts to the Netty framework[2] to support inter-process communication through sockets (although it was designed to hide this from the programmer). The framework will be discussed in the labs, and example protocols made available to students.
The javadoc of the framework can be (temporarily) found here: `http://asc.di.fct.unl.pt/~jleitao/babel/doc`.
The framework was specifically designed thinking about teaching distributed algorithms in advanced courses. A significant effort was made to make it such that the code maps closely to protocols descriptions using (modern) pseudo-code. The goal is that you can focus on the key aspects of the protocols, their operation, and their correctness, and that you can easily implement and execute such protocols.
The course discussion board in Moodle can (and should be) used to ask for support in using or clarify any aspect on its usage.
This is a prototype framework, and naturally some bugs (or idiotic interfaces) can be found. We will address these as soon as possible. The development team is open to suggestions and comments. You can make such comments or improvement suggestions either through the Moodle discussion board or by e-mail to the course professor.

## 4 Operational Aspects and Delivery Rules

### *Group Formation*

The groups should be the same as in phase 2.

[1] `http://nova-lincs.di.fct.unl.pt`
[2] `https://netty.io`

## *Evaluation Criteria*

The project delivery includes both the code and a written report that must contain clear and readable pseudo-code for each of the implemented protocol/client application, alongside a description of the intuition of the protocol. A correctness argument for each layer will be positively considered in grading the project. The written report should provide information about all experimental work conducted by the students to evaluate their solution in practice (i.e., description of experiments, setup, parameters) as well as the results and a discussion of those results.
The project will be evaluated by the correctness of the implemented solutions, its efficiency, and the quality of the implementation. With relative weights of, respectively: $50\%$, $35\%$, and $15\%$.
The quality and clearness of the report of the project will impact the final grade in $10\%$, however the students should notice that a poor report might have a significant impact on the evaluation of the correctness the solutions employed (which weight $50\%$ of the evaluation for each component of the solution).
This phase of the project will be graded in a scale from 1 to 20. With the following distribution of weights[3]:

Developed Distributed Protocols: 12/20

Test Application and Evaluation: 5/20.

Written Report: 4/20

## *Deadline*

Delivery of phase 3 of the project is due on 6 December 2019 at 23:59:59.
The delivery should be made by e-mail to the address jc.leitao@fct.unl.pt with a subject: "ASD Phase 3 Delivery - #student1 .. #studentn".
The e-mail should contain two attachments: $i)$ a zip file with the project files, without libraries or build directories (include src, pom.xml, and any configuration file that you created) and $ii)$ a pdf file with your written report. If you have an issue with sending the zip file you can do one of the following: $a)$ put a password on the zip file that should be 'asd2019' with no quotation marks; or $b)$ put the zip file in a google drive, and send the public access link.
Project deliveries that do not follow these guide lines precisely may not be evaluated (yielding an automatic grade of zero).

**Extra Days Policy** While the official delivery date is the one reported above. Students can use up to $5$ extra days to perform this delivery. Each extra day will incur in a penalty of $0.2$ values in the grade of this phase of the project (before applying the obvious rule of max(Grade, 20) since each phase of the project can be graded to at most $21$ values.

[3](Yes, the sum of all components is not 20)