# Algoritmos e Sistemas Distribuídos- Project Phase 2

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 $\alpha$

$11^{th}$ October 2019

## 1 Overview

This document discusses the second 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.

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 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).

## 2 Solution Architecture

Figure 1 illustrates the proposed layering of protocols to solve the phase 2 of the project. The figure also describes the interactions between the new protocols to add in this phase, from which the **interfaces** of the protocols can be (partially) derived.

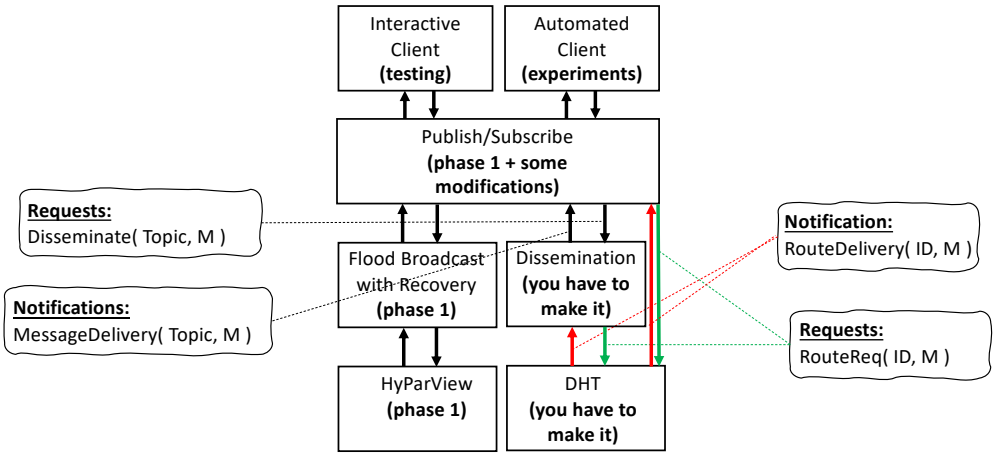


Figure 1: Proposed (abstract) composition of protocols to solve the project.

The key goals of this phase of the project is two fold. First, you should define a scheme where each topic becomes managed by one of the processes in the system, notice that the load should be distributed as uniformly as possible across all nodes in the system. Second, we aim at improving the dissemination of messages for topics. In particular, topics that have few elements should strive to minimize the number of processes that have to collaborate in the dissemination process, effectively reducing the overhead in the dissemination process. To assist you in this, you should make sure that every message published in a topic is first sent (by the Pub-Sub protocol) to the manager of that topic, and then that manager will decide how to and disseminate the message throughout subscribers.
In the following some details are provided related with each of the (novel or to be modified) protocols.

## *2.1 DHT Protocol*

The DHT will allow you to primarily route messages among nodes using some application level addresses. This can be useful in the context of the project to assist in determining who should be responsible for each topic and potentially to route messages among nodes with a lower message cost. An interesting starting point is the Chord protocol [2].

**Requests:**

**ROUTEREQ:** This request allows any protocol to request the DHT to route a message to a given node identified by ID or whose ID is closest to the indicated ID (ID is probably not an IP address). M should be encapsulated into a notification that will be consumed by the protocol that should receive the message when it arrives at the destination.

**Notification:**

**ROUTEDELIVERY:** This is a notification that indicates the reception of a message that is not itself a notification. This is only triggered when the DHT has no better (or closest) candidate to forward the message to.

**Optional:**

It might be useful to have an additional notification issued by the DHT protocol to indicate that a message is being routed through this node and hold the routing process until another request is received. This notification is only issued in nodes along the route of the message and not the end destination. This might become more clear if you check the Scribe paper [1].

### *2.2 Dissemination Protocol*

This is a protocol that will make message dissemination for particular topics, in an efficient way. The protocol design can be selected on your own. It is proposed that you operate on top of the DHT but you might decide to run it on top of HyParView.

**Requests:**

**DISSEMINATE({TOPIC, MESSAGE}):** This request is exposed to other protocols to reliably disseminate a given message to all processes in the system that are interested in Topic.

**Notifications:**

**MESSAGEDELIVERY({TOPIC, MESSAGE}):** This is a notification that carries the contents of messages received at the local node (it might happen that the local node is not interested in this Topic. That filtering is handled by the Publish/Subscribe layer).

### *2.3 Publish/Subscribe Protocol*

The publish subscribe has the same interface as in phase 1. However, the protocol now has access to more than one communication protocol stack, and can employ different dissemination strategies. Note that the subscribe and unsubscribe mechanisms might no longer be contained within this protocol (i.e., to process these requests, interactions with other protocols or processes might be required).

## 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.

[1] `http://nova-lincs.di.fct.unl.pt`

[2] `https://netty.io`

# 4 Operational Aspects and Delivery Rules

### *Group Formation*

The groups should be the same as in phase 1.

### *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 1 of the project is due on 10 November 2019 at 23:59:59.
The delivery should be made by e-mail to the address jleitao@fct.unl.pt with a subject: "ASD Phase 2 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).

# References

[1] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, NGC '01, pages 30–43, London, UK, UK, 2001. Springer-Verlag.

[2] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

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