Algoritmos e Sistemas Distribuídos- Project Phase 1

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 2.0

20th September 2018 (Revised October 10th)

1 Overview

This document discusses the first 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 1 of the project[1]. The figure also describes the interactions between the protocols, from which the **interfaces** of the protocols can be (partially) derived.

[1]While the layering is proposed, from the perspective of evaluation it is *mandatory*.

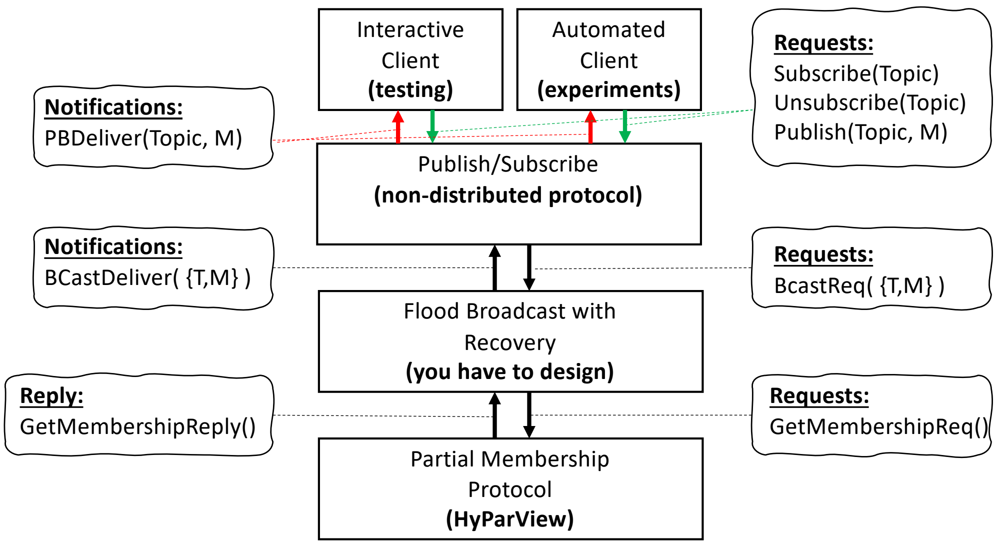
Interactive Client (testing)
Automated Client (experiments)
Requests: Subscribe(Topic) Unsubscribe(Topic) Publish(Topic, M)
Notifications: PBDeliver(Topic, M)
Publish/Subscribe (non-distributed protocol)
Notifications: BCastDeliver({T,M})
Requests: BcastReq({T,M})
Flood Broadcast with Recovery (you have to design)
Reply: GetMembershipReply()
Requests: GetMembershipReq()
Partial Membership Protocol (HyParView)

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

In a nutshell, and using a bottom-up approach, the system to be developed in this phase of the project will use an *unstructured overlay network* that is popular for its robustness to faults called *Hybrid Partial View* or simply, HyParView [1]. This layer is be responsible for managing the membership of the system. It offers to the layers above a partial view of the processes currently active.
The following layer is responsible for disseminating messages, reliably, throughout all (correct) processes in the system. This layer will be designed by the students and it **should** combine different flavors of Gossip to ensure the reliability of the dissemination process.
On top of the dissemination layer, a (thin) protocol will manage the subscriptions of applications (denoted in the Figure as clients). Based on that (local) information, that protocol is responsible for deciding when a message being disseminated throughout the network should be delivered to the application. This layer is also responsible for handling (explicit) requests from applications to disseminate new messages throughout the system. Notice that this protocol, in this phase, is not distributed. Essetially it operates as a filter for broadcasted messages, ensuring that only relevant messages for applications are delivered to them.
Finally, the clients interact with the publish/subscribe protocol to subscribe/unsubscribe from topics, and to publish and receive messages from the system. The goal here is to build two different (client) applications to the publish/subscribe protocol: i) an interactive client that receives user input from the command line. This client will essentially be used for demonstrating the results of the project and for testing; and ii) an automated client that will automatically generate user actions (subscribe/unsubscribe/publish) and record all activities (and received messages) for post-mortem analysis. This client will be used to perform a simple experimental benchmark, whose results should be reported with the phase 1 delivery (and compared with experimental work to be conducted in the following phases).
In the following a more detailed explanation of the interface of each protocol will be provided. Notice that this discussion only focus on the interface that is used by other protocols, and therefore is incomplete, since protocols also generate and consume messages, and might need to resort to timers to manage periodic or timed actions.

2.1 Partial Membership Protocol: HyParView

Requests:

GETMEMBERSHIPREQUEST: This request has the goal of exposing the contents of the *active partial view* maintained locally by the HyParView protocol.

Replies:

GETMEMBERSHIPREPLY: This is a reply to the GETMEMBERSHIPREQUEST. It reports to the requester a Set containing the identifiers (typically a data structure containing the IP address and listen port of a process) of all nodes in the active view of HyParView when the request was processed (i.e., not a reference to the internal state of the protocol).

Alternative:

Students that wish to do so, can replace the proposed interface of the HyParView protocol by a set of notifications (NEIGHBORUP and NEIGHBORDOWN) that, in a reactive and asynchronous fashion, report to the protocol above changes in the contents of the local active view of the protocol.

2.2 Flood Broadcast Protocol with Recovery

Requests:

BCASTREQUEST({TOPIC, MESSAGE}): This request is exposed to other protocols to reliably broadcast a given message to all processes in the system (using a gossip approach). Messages disseminated in this fashion are: i) delivered to all processes including the one that made the original request; and ii) delivered to all processes independently of transient failures that might lead a process to, for a small amount of time, become disconnected from all other correct processes. While this protocol should be generic and easily reused in other application contexts or protocol stacks, in the context of this phase of the project, messages being disseminated will contain both a Topic and the message generated by the application layer. Notice, that this request does not have a reply associated.

Notifications:

BCASTDELIVER({TOPIC, MESSAGE}): This is a notification that carries the contents of messages being broadcast throughout the system. The contents of the message should be presented to the above layers exactly in the format in which they were provided to this protocol through the BCastRequest interface. Notice, that no duplicate should be delivered. This implies that it is the responsibility of the protocol to filter potential duplicates received from the network.

2.3 Publish/Subscribe Protocol

Requests:

SUBSCRIBE(TOPIC): This request notifies the system that the requester is interested in receiving (through a notification) all messages published that are associated with the topic TOPIC. This request does not have a reply associated. This request has no effect if the provided topic was already subscribed.

UNSUBSCRIBE(TOPIC): This request notifies the system that the requester is no longer interested in receiving messages published on topic TOPIC. This request does not have a reply associated. This request has no effect if the provided topic was not subscribed.

PUBLISH(TOPIC, MESSAGE): This request publishes to the system a message with contents MESSAGE tagged with TOPIC. This request does not have a reply associated. Notice that a client/application is not required to be subscribed to a topic to publish messages tagged with that topic. Multiple requests with the same contents should be considered different messages.

Notifications:

PUBSUBDELIVER(TOPIC, MESSAGE): This notification is generated to protocols/applications that have subscribed to a Topic when a new message (for that topic) becomes available. Each (unique) message should be delivered a single time.

2.4 Interactive Client

The interface exposed by the interactive client for users is based on the (terminal) command line. Only a single operation is processed at a time. Operations map directly the request operations exposed by the Publish/Subscribe Protocol. Examples of commands are provided below:

```
subscribe di-fct

unsubscribe di-fct

publish di-fct The best department in the World.
```

The application should also process messages delivered to it by the underlying publish/subscribe protocol. This should generate an output similar to the following (considering the example above):

```
received event: Topic: di-fct Message: The best department in the World.
```

Students can improve this interface if they want.

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[2] written by João Leitão, Pedro Fouto, and Pedro Ákos Costa, whose internal code-name is **Babel**.
The framework resorts to the Netty framework[3] 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.

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

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

4 Operational Aspects and Delivery Rules

Group Formation

Students can form groups of up to three students to conduct the project. Groups composition cannot change across the different phased of the project. While students can do the project alone or in groups of two students this is **highly discouraged**, as the load of the project was designed for three people.

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 (except for HyParView which is a well known 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[4]:

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 15 October 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 1 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] J. Leitão, J. Pereira, and L. Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 419–429, June 2007.

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