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 0.5

October 2, 2024

1 Overview

This document discusses the first phase of the ASD project for the 2024/25 edition. The project has two phases which are fully independent between them. The primary goal of the first phase of the project is to build a robust and efficient peer-to-peer point-to-point communication system and experiment it in practice. In this context, the point-to-point communication system will allow peers in the system to establish direct communication channels between them and exchange messages. Peers are identified by a unique random identifier (i.e., a large bit string) instead of IPs and ports. Hence when the application (executing at a given peer) desires to send a message to another peer, it will first determine the ip and port of the node with whom it should communicate, after this step a TCP connection should be established and the message sent. To this end the point-to-point communication protocol will rely on a distributed hash table (DHT) to be able to translate the unique random identifier into the current IP address and port where the corresponding peer can be reached.

While the primary goal described above is already interesting, one has to question himself of what should happen when a peer desires to send a message to another peer when it is not currently online. One way would be for the peer that wants to send the message to store it locally at the point-to-point communication protocol, and periodically to retry to locate the destination peer and deliver the message. While this would be a viable approach, it should be noted that this could lead to the message to not be delivered if the sender becomes offline. Instead, in addition to this strategy, we will complement it by also delivering the message to another peer (call it a helper node), whose identifier is closer to the real destination, that will also attempt to deliver the message in the name of the sender. We will again use the DHT to identify the helper node. Evidently, while this mechanism will improve the robustness of the point-to-point communication system and allow communication among peers in a way that is decoupled in time, it also creates the possibility for a receiver to get the same message more than once. This should be evidently avoided. Also, students should consider what happens if the helper node becomes unavailable. In this case the delivery of a pending message becomes again fully dependent on the original sender, which is clearly undesirable.

Naturally, and in line with most peer-to-peer systems, every node in the system can simultaneously act as a sender and a receiver. Also a single node might have multiple messages to be sent to the same or different destinations before previous messages are delivered. A final note, students can assume that the random unique identifier of a node will

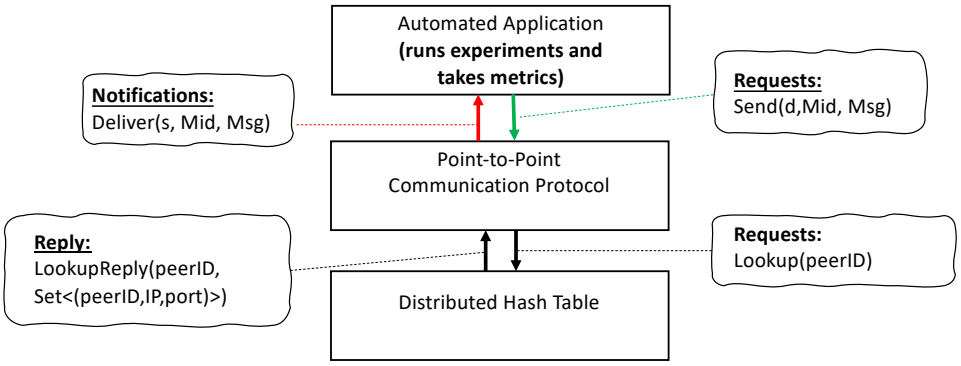

Figure 1: Architecture of a Process.

never change, whereas the IP address and port can change over time (i.e., because a user turns off his laptop, moves to a different location, and turns his laptop on).
Students should present their solution as a set of three protocols, the application that is responsible for sending and receiving messages (and eventually interact with a human user), the point-to-point communication protocol that is responsible for making sure that messages sent by the application are delivered to the correct destinations (including acting as helper nodes for other senders), and a distributed hash table protocol that will allow to perform application-level routing. The proposed solution should be evaluated, in a way that shows both the effectiveness and the cost of the proposed solution. An initial version of the application will be provided to students at a later date (at most two days after the publication of this document in clip).
In summary, the project is designed to allow students to: i) implement different types of decentralized distributed protocols; ii) understand trade-offs among different points in the design space; iii) think and design solutions that are both correct and efficient; iv) conduct experimental assessment of their ideas.
In the following the target protocol architecture (and some of their interfaces) that should be constructed for this project are 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 operational aspects and delivery rules for this phase of the project (Section 4).

2 Solution Architecture

Figure 1 illustrates the layering of protocols that you will have to explore during the execution of the first phase of the project[1]. The figure also describes the interactions between the protocols, from which the **interfaces** of the protocols can be (directly) derived.
The project will have a strong emphasis on the experimental component, where you will assert the performance of your solution with at least 100 different processes. To do this you will be responsible for implementing one distributed hash table based on solutions provided in scientific literature, and the point-to-point communication protocol as described above that should exploit the interface and functionality of the DHT protocol implemented by the group.
Notice that the interface to which we refer above is restricted to the events that are used by protocols executing within the same process to interact among them (which will be materialized by Requests and Replies, as well as Notifications). The messages exchanged by each protocol that you develop with the (equivalent) protocol on other processes is not fixed. You should model these messages following the specification of the protocol that you are implementing. Notice also that messages will always be processed by the same protocol (albeit across different processes).
Below the options for the different hash tables that you can consider.

[1] for evaluation purposes this layering is *mandatory*.

2.1 Distributed Hash Tables (Structured Overlay Network)

You can pick one of the following two distributed hash tables for this phase of the project:

Chord: Chord [2] is potentially the most well known DHT protocol in the world. The intuition behind Chord is quite simple, nodes form a ring that is ordered according to the random identifiers of nodes (where the node with the highest identifier is connected to the one with the lowerst identifier to ensure the ring topology). For fault-tolerance each node periodically checks with the node in front of him to: i) make sure that the node in front of im believes that the local node is his predecessor (and fix the situation if this is not true) and ii) to gather information about other nodes in front of him such that if its sucessor fails it can connect to the node in front of that one. For efficiency, each node also keeps direct links to increasingly distant nodes in the ring in front of him, as to speed up the routing of requests.

Kademlia: Kademlia [1] is a very popular DHT that has been used in production in the context of several systems, such as bit-torrent clients (as a complementary mechanism to locate additional sources for torrents) and in the Interplanetary file system (IPFS) as the underlying application level routing protocol. Kademlia works in a different way to Chord. Instead of a ring the protocol, in some sense, defines a ordered tree of nodes in the system where each node is a k-bucket containing up to k different nodes in that segment of the network. The protocol monitors all messages passing through it (i.e., routing requests) to become aware of other nodes in the system as to fill local k-buckets. When the k-bucket where the local node is fills, it is divided. Other k-bucktes have a fixed size k that is never surpassed.

2.2 Applications

The applications will be provided to you (both source code and maven project will be available on clip before the 6^{th} of October). The applications are relatively simple but provide a set of features that will be quite useful for you.
The application will generate requests automatically and periodically to send messages to a set of peer identifiers extracted from a configuration file. The application registers information about messages received and sent for from or to different peer identifiers. This will greatly simplify the collection of performance metrics during the evaluation work.
There will be two variants of the application, one is an interactive one and will allow you to do debug, by generating requests to send requests to specific peer ids. The second will be an automated client application that you will be using for conducting large scale experiments (as it would be unfeasible for you to be issuing text commands to one hundred different processes simultaneously).
Furthermore, the automated client application allows you to configure several operational aspects for your experiments. The two key parameters that we will be interested in manipulating for experiments are:

Rate of Request: The time interval between the generation of requests by the application.

Payload size: The size of the content being sent by the application.

This section will be revised in a future release of this document - if required - with more details of the provided application.

2.3 Evaluation

You will conduct an experimental evaluation where you will assert the performance and cost of your proposed solution, ideally considering different systems size (i.e., number of nodes) to study the scalability of the solution, and different parameters associated with the application workload as described above. You will use the computational cluster of DI/NOVA LINCS (see below) to conduct these experiments.
You should compare several aspects of the operation of your solution. In particular you must evaluate the following aspects (you can evaluate others that you consider relevant):

Reliability: The reliability is defined as the fraction of messages received by (correct) processes.

Live Latency: The time required for delivering a message to a given peer when both the sender and receiver are active at the time of the message transmission.

Indirect Latency: The time required a receiver to effectively receive a message sent while he was offline after he joins the system.

Redundancy: The total number of messages received by processes at the point-to-point communication protocol divided by the total number of messages requested to be sent by the application.

Total Messages/Bytes Transmitted: The amount of messages/bytes sent to the network collectively by all processes during the experiment.

Total Messages/Bytes Received: The amount of messages/bytes received from the network collectively by all processes during the experiment. This number should be the same as the above in the case where no messages were lost of processes failed.

In your experiments you can use two different values for the payload size (one small another bigger) and two different values for the rate of transmission (again, one small and another bigger). You can conduct experiments only in steady state (i.e., in scenarios where no process crashes during the experiment) but students are highly encouraged to design experiments where nodes might crash of join the system while the system is already operating.
You evidently can consider other experimental conditions **in addition** to the ones described above. These can include varying parameters that govern the workload of the system, or different parameters for the point-to-point communication protocol or the DHT that you selected. Every extra aspect will be taken into consideration in the evaluation process. However, students are not recommended to try every possible combination.

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 Pedro Fouto, Pedro Ákos Cost, João Leitãoa, named **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 was already presented and discussed in the labs, and example protocols have been made available to students. Moreover the top layer that is responsible for injecting load in the system (i.e., propagate messages and receive them) is provided by the faculty.
The javadoc of the framework can be found here: `https://asc.di.fct.unl.pt/~jleitao/babel/`.
The framework was specifically designed thinking about two complementary goals: i) quick design and implementation of efficient distributed protocols; and ii) 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.

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 phases 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.

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

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

Since you will be given access to the DI and NOVA LINCS research cluster (see below) to conduct your experiments and extract performance numbers, you must pre-register your group as soon as possible such that you can get credentials to use the cluster. This can be done by sending an e-mail to the course professor with subject *ASD 24/25 GROUP REGISTRATION*. The e-mail should contain for each member of the group: student number, full name, institutional e-mail. Even students that plan to do the project alone must register their group. You will get a reply from the professor (eventually) providing your user name and password to access the cluster. **This e-mail should be sent by October** 8^{th}.

The DI and NOVA LINCS research cluster

The cluster is used for (mostly) research purposes, although some (advanced) courses also use it. The cluster has a limited amount of machines, and hence people might need to compete over computation time. The cluster features a reservation mechanism where you can reserve a set of machines for exclusive access during a period of time. You should not use the cluster for development purposes and only to extract final numbers. Moreover, each group should only reserve at most two machines at a time. Each machine should be able to execute several tens of independent java processes that will act as different nodes. Docker is available in the cluster but you should not need to use it. We will provide simple scripts to help you in executing your experiments. Finally, each group will be restricted to use only $3h \times 2$ of computational time. This should allow you to execute your experiments for 3 hours on two machines. Notice that automation is key for success, and that should allow you to run experiments during the night period (where typically usage of the cluster is much more reduced).
The technical specification of the cluster as well as the documentation on how to use it (including changing your group password and making reservations) is online at: `https://cluster.di.fct.unl.pt`. You should read the documentation carefully. The cluster is accessible from anywhere in the world through ssh. Be careful with password management. Never use a weak password to avoid attacks and intrusions on our infrastructure. Incorrect or irresponsible use of the department resources made available to students will be persecuted through disciplinary actions predicted in the School regulations.

Evaluation Criteria

The project delivery includes both the code and a written report that should have the format of a short paper. Considering that format, this document referes from this point onward as simply *paper*.
The paper should be at most 10 pages long, including all tables, figures, and references. It should be written with font size no bigger than 10pts and be in two column format. It is highly recommended (for your own sanity) that you use LaTeXto do this. If you do not know LaTeXit is indeed a great time to learn. A LaTeXtemplate for writing the paper shall also be provided. The course professor will also dedicate an entire lab class on how to structure and write a paper.
The paper must contain clear and readable pseudo-code for each of the implemented protocols, alongside a description of the intuition of these protocols. A correctness argument for protocol that was devised or adapted by students will be positively considered in grading the project. The written report should also 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 implementations (in terms of code readability).
The quality and clearness of the report of the project will have an impact the final grade. Notice however that students with a poorly written report might (accidentally) be penalized on the evaluation of the correctness the solutions employed.
This phase of the project will be graded in a scale from 1 to 20 with the following considerations:

Specification of the protocols (and ideas employed for the point-to-point communication protocol): 7 values

Quality and Correctness of the Implementation: 7 values

Experimental Evaluation: 7 values

Deadline

Delivery of phase 1 of the project is due on October 20^{th} 2024 at 23:59:59.
The delivery shall be made by e-mail to the address jc.leitao@fct.unl.pt with a subject: "ASD 24/25 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 'asd2024' 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 guidelines precisely may not be evaluated (yielding an automatic grade of zero).

References

[1] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[2] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on networking*, 11(1):17–32, 2003.