Lab Exercise 3
January 2, 2018
IN4150 Distributed Algorithms

4501667 - A.D. Dimitrova
4503813 - L.J.R. Weijs
Group 7

Distributed Algorithms

Professor: D.H.J. Epema

TU Delft

1 Assignment

In the current assignment we implement a randomized algorithm for Byzantine agreement, Ben-Or's algorithm for synchronous and asynchronous systems.

2 Decisions

For the implementation we made several architectural and decisions and assumptions. We store all information on an agent (Byzantine) into a node class like stated in Figure 1. Furthermore the factory design pattern is included to be able to talk between different agents via Java RMI. Also in our case we assumed that the agents need to agree on a binary value (0 or 1) in the system, when agents do not know what to send they will send a value of -1 (”I don't know”, as per the algorithm description). We designed several test cases once we confirmed our algorithm implementation works and the initial values for the Byzantine nodes are given for each specific test case in the next section.

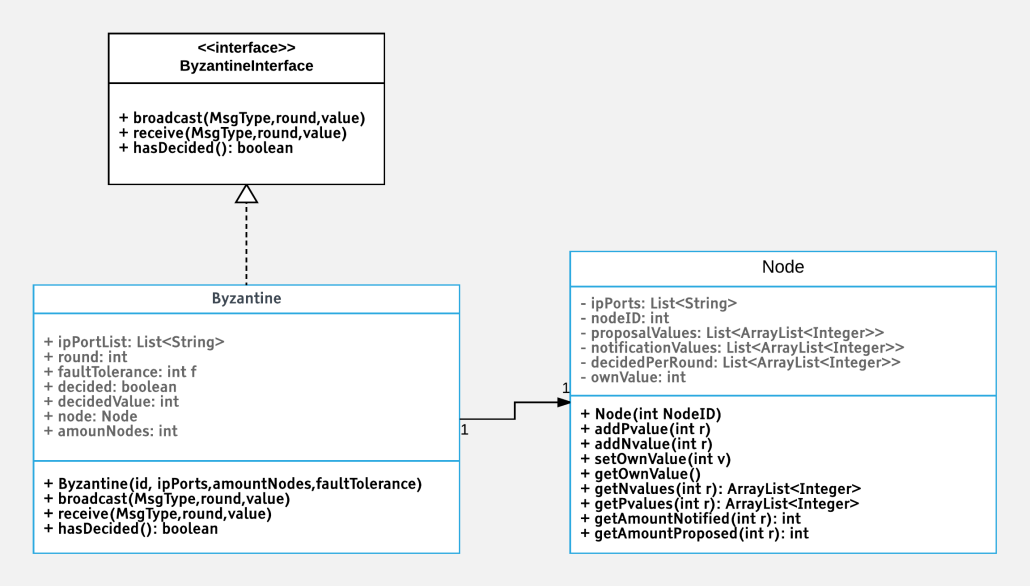

Figure 1 – Class Diagram

3 Test Cases

In this section we describe the different test cases in order to find the limits of the Ben-Or algorithm implemented by us. The different test cases are visualized in Figure 2. The test cases are taken into consideration into their respective order.

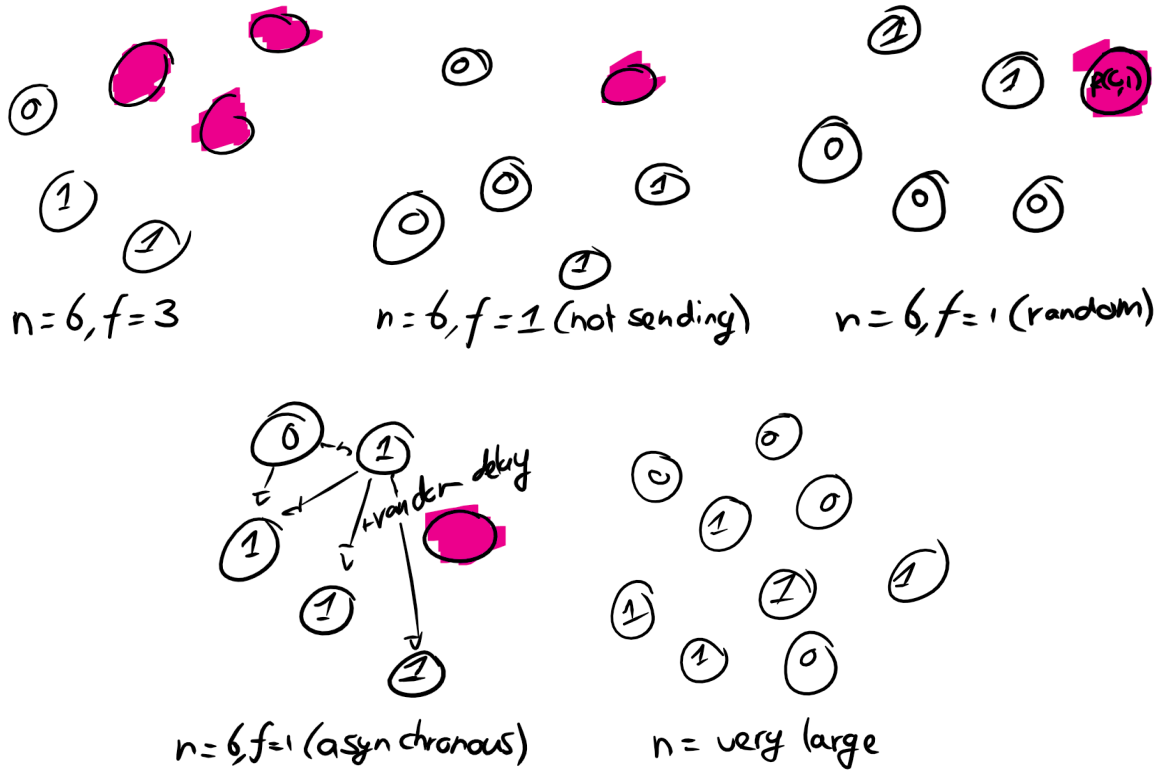

Figure 2 – Five test cases in order to test the Ben-Or Consensus Algorithm, n represents the amount of nodes in the system and f represents the amount of faulty nodes in the system also denoted by their purple color.

3.1 $n = 6, f = 3$

In this test case the amount of faulty nodes is larger than the necessary condition of $n > 5 \times f$. This will mean that the loyal nodes are not guaranteed to find consensus, will it is possible due to the randomization of the algorithm. Even though the faulty processes broadcast faulty messages, random or the opposite of what is to be expected, the algorithm will never finish due to the constraint that the loyal processes wait for $3 \times f$ proposal values before they decide. Because there are not even nine processes in this problem statement the algorithm will continue to wait forever for proposal values.

3.2 $n = 6, f = 1$ (not sending)

In this subsection we consider a single faulty process which holds the necessary condition of $n > 5 \times f$, and does not send anything. Please note that the starting values of the nodes are shown in Figure 2. The following results were generated by the algorithm, with consensus already after the first round:

<< DECIDED >> byzantine 0, value: 0 After round: 1
<< DECIDED >> byzantine (traitor) 1, value: 0 After round: 1

<< DECIDED >> byzantine 2, value: 0 After round: 1
<< DECIDED >> byzantine 3, value: 0 After round: 1
<< DECIDED >> byzantine 4, value: 0 After round: 1
<< DECIDED >> byzantine 5, value: 0 After round: 1

The results show that the algorithm is able to correctly handle this test case because all the non-faulty nodes decide on the same value.

3.3 $n = 6, f = 1$ (random)

In this next subsection we consider the same amount of nodes and faulty nodes but now with a faulty node sending random broadcasts. And node here that even though the amount of zeros are in the majority, it does not mean per se that the nodes decide on 0. This is due to the fact that in the first round some nodes just do not know what to propose. Because of the many "I do not know"s proposal values the nodes are not able to decide and therefore flip a coin on their own value, this could go either way to zero or to one and even has the possibility to loop endlessly. The results are given below,

<< DECIDED >> byzantine 0, value: 1 After round: 4
<< DECIDED >> byzantine 1, value: 1 After round: 4
<< DECIDED >> byzantine 2, value: 1 After round: 4
<< DECIDED >> byzantine 3, value: 1 After round: 4
<< DECIDED >> byzantine 4, value: 1 After round: 4
<< DECIDED >> byzantine (traitor) 5, value: 1 After round: 4

3.4 $n = 6, f = 1$ (asynchronous)

This algorithm is also able to even the possibility of an asynchronous property of the system. For this we introduced random delay between the broadcasting of the messages from 0 to 2 seconds. Again the starting values are stated in the fourth sub-figure of Figure 2 and there is a faulty process which sends random messages. The results are shown below,

<< DECIDED >> byzantine 5, value: 1 After round: 2
<< DECIDED >> byzantine 0, value: 1 After round: 4
<< DECIDED >> byzantine 1, value: 1 After round: 4
<< DECIDED >> byzantine 2, value: 1 After round: 4
<< DECIDED >> byzantine (traitor) 3, value: 1 After round: 4
<< DECIDED >> byzantine 4, value: 1 After round: 4

Even though the round of the parties decided are not the same they reached the same consensus. After a node has decided it will still keep on sending its own value as a notification and participate in the proposal phase even though it has decided already.

3.5 $n =$ very large

In this last test case we tested how many nodes we were able to host in a single machine. The binding of the byzantines/nodes on the ports of the machine is not limiting factor at all, we tested for 1000 nodes to be binded to the ports and although it took a while the machine was able to handle this. And finding port which are not already occupied seemed to be larger problem due to the various processes on the computer. Because of this tedious process of selecting ports which

were not in use, we assumed the algorithm is able to run as long as it gets enough time and enough ports to occupy.

3.6 Extra test case: n = 75

In this test case we take 75 nodes of which 14 are traitors and which send nothing, 6 have an initial value of 1 and the rest, 55, have initial values of 0. After a single round they will decide on 1. The results are printed below:

ID: 0— Traitor: true — Type: N
ID: 1— Traitor: true — Type: N
ID: 2— Traitor: true — Type: N
ID: 3— Traitor: true — Type: N
ID: 4— Traitor: true — Type: N
ID: 5— Traitor: true — Type: N
ID: 6— Traitor: true — Type: N
ID: 7— Traitor: true — Type: N
ID: 8— Traitor: true — Type: N
ID: 9— Traitor: true — Type: N
ID: 10— Traitor: true — Type: N
ID: 11— Traitor: true — Type: N
ID: 12— Traitor: true — Type: N
ID: 13— Traitor: true — Type: N
<< DECIDED >> byzantine 14, value: 0 After round: 1
<< DECIDED >> byzantine 15, value: 0 After round: 1
<< DECIDED >> byzantine 16, value: 0 After round: 1
<< DECIDED >> byzantine 17, value: 0 After round: 1
<< DECIDED >> byzantine 18, value: 0 After round: 1
<< DECIDED >> byzantine 19, value: 0 After round: 1
<< DECIDED >> byzantine 20, value: 0 After round: 1
<< DECIDED >> byzantine 21, value: 0 After round: 1
<< DECIDED >> byzantine 22, value: 0 After round: 1
<< DECIDED >> byzantine 23, value: 0 After round: 1
<< DECIDED >> byzantine 24, value: 0 After round: 1
<< DECIDED >> byzantine 25, value: 0 After round: 1
<< DECIDED >> byzantine 26, value: 0 After round: 1
<< DECIDED >> byzantine 27, value: 0 After round: 1
<< DECIDED >> byzantine 28, value: 0 After round: 1
<< DECIDED >> byzantine 29, value: 0 After round: 1
<< DECIDED >> byzantine 30, value: 0 After round: 1
<< DECIDED >> byzantine 31, value: 0 After round: 1
<< DECIDED >> byzantine 32, value: 0 After round: 1
<< DECIDED >> byzantine 33, value: 0 After round: 1
<< DECIDED >> byzantine 34, value: 0 After round: 1
<< DECIDED >> byzantine 35, value: 0 After round: 1
<< DECIDED >> byzantine 36, value: 0 After round: 1
<< DECIDED >> byzantine 37, value: 0 After round: 1
<< DECIDED >> byzantine 38, value: 0 After round: 1
<< DECIDED >> byzantine 39, value: 0 After round: 1
<< DECIDED >> byzantine 40, value: 0 After round: 1
<< DECIDED >> byzantine 41, value: 0 After round: 1
<< DECIDED >> byzantine 42, value: 0 After round: 1

<< DECIDED >> byzantine 43, value: 0 After round: 1
<< DECIDED >> byzantine 44, value: 0 After round: 1
<< DECIDED >> byzantine 45, value: 0 After round: 1
<< DECIDED >> byzantine 46, value: 0 After round: 1
<< DECIDED >> byzantine 47, value: 0 After round: 1
<< DECIDED >> byzantine 48, value: 0 After round: 1
<< DECIDED >> byzantine 49, value: 0 After round: 1
<< DECIDED >> byzantine 50, value: 0 After round: 1
<< DECIDED >> byzantine 51, value: 0 After round: 1
<< DECIDED >> byzantine 52, value: 0 After round: 1
<< DECIDED >> byzantine 53, value: 0 After round: 1
<< DECIDED >> byzantine 54, value: 0 After round: 1
<< DECIDED >> byzantine 55, value: 0 After round: 1
<< DECIDED >> byzantine 56, value: 0 After round: 1
<< DECIDED >> byzantine 57, value: 0 After round: 1
<< DECIDED >> byzantine 58, value: 0 After round: 1
<< DECIDED >> byzantine 59, value: 0 After round: 1
<< DECIDED >> byzantine 60, value: 0 After round: 1
<< DECIDED >> byzantine 61, value: 0 After round: 1
<< DECIDED >> byzantine 62, value: 0 After round: 1
<< DECIDED >> byzantine 63, value: 0 After round: 1
<< DECIDED >> byzantine 64, value: 0 After round: 1
<< DECIDED >> byzantine 65, value: 0 After round: 1
<< DECIDED >> byzantine 66, value: 0 After round: 1
<< DECIDED >> byzantine 67, value: 0 After round: 1
<< DECIDED >> byzantine 68, value: 0 After round: 1
<< DECIDED >> byzantine 69, value: 0 After round: 1
<< DECIDED >> byzantine 70, value: 0 After round: 1
<< DECIDED >> byzantine 71, value: 0 After round: 1
<< DECIDED >> byzantine 72, value: 0 After round: 1
<< DECIDED >> byzantine 73, value: 0 After round: 1
<< DECIDED >> byzantine 74, value: 0 After round: 1

4 Discussion

Our default test case to prove the algorithm works, was with 6 nodes, 1 out of which is faulty ($n = 6, f = 1$) and sends random value. This experiment is not described, but shown to the Teaching assistants as a regular implementation before testing out with more customized cases. This default setting satisfies the requirement for the number of traitors to be $1/5$th of the total amount of nodes in the system in order for the system to be able to reach consensus. We prove the algorithm works by testing in a few more settings. Section 3.1 is interesting due to the fact that all processes wait, and therefore simulates an asynchronous system, for certain amount of messages ($3\text{x}f$) in order to decide, which does not happen and hence the algorithm will never reach consensus in favor of the loyal nodes. We wanted to check our computers' performance, so we tested a few cases with 75 nodes or more. They hold the same result but take up more space, hence we show only one of them in the last section of the report.