



# TÉCNICO LISBOA

## Fault-Tolerant Distributed Systems

2º Semester – Academic Year 2013/2014

---

David Daharewa Gureya, 79533 - [david.gureya@tecnico.ulisboa.pt](mailto:david.gureya@tecnico.ulisboa.pt)

Kuganesan Srijeeyanthan, 79531 - [kuganesan.srijeeyanthan@ist.utl.pt](mailto:kuganesan.srijeeyanthan@ist.utl.pt)

Gayana Ranganatha Chandrasekara Pilana Withanage, 79529 - [gayana.withanage@ist.utl.pt](mailto:gayana.withanage@ist.utl.pt)

### Randomized Consensus Algorithm Implementation

---

## Introduction

This report is presented in order to provide a detailed description with practical implementation results, of the project we are assigned which was “Randomized Binary Consensus” and the extended version of it, “Randomized Consensus with Large Domain”. We used the Appia framework which is an event based layered protocol stack in order to implement algorithm. The report consists several section where section 1 we discuss the structure of the stack, and in section 2 explains the implementation while section 3 will describe the evaluation.

## Section 1: Protocol Stack

In order to implement the Randomized consensus, we require the regular and uniform reliable broadcast mechanisms as prerequisite components. Therefore, we use Best Effort Reliable Broadcast and Eager Reliable Broadcast respectively. Below figure 1 depicts the protocol stack we developed.

### Application Layer

This is the top most layer where the user interacts with, and consensus algorithm is triggered. After the successful completion of a consensus instance the decision also delivered to this layer from the below layers.

## Consensus Layer

This layer consists of the protocol implementation for Randomized Binary Consensus and the extension for Large Domain. Consensus value proposal, condition verification, phase definition, decision delivery are the key functionalities of this layer.

## Eager Reliable Broadcast Layer

Here, the messages it receives are routed in a uniform reliable broadcast fashion so that an undelivered message is delivered instantly and a clone of that message will be pushed downwards for broadcast again.

## Best Effort Broadcast Layer

This layer contains the protocol to route messages to each correct process and to provide regular reliable broadcast capability.

## TCP Complete Layer

This layer assures that the messages are routed via TCP channel where the FIFO order is guaranteed.

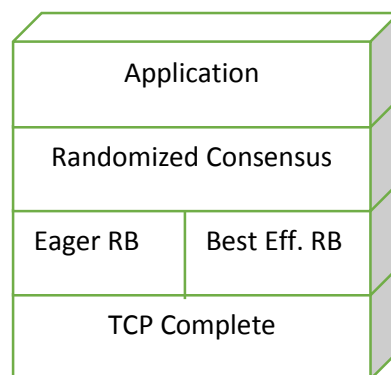


Figure 1: Protocol stack for randomize consensus implementation

## Section 2: Implementation

During the initial cycle of implementation we focused on developing the Randomized Binary Consensus as described in the course text book. Therefore, we developed the top four layers shown in figure 1. Application initialize the processors reading a configuration file which contains the processors that are going to be participated for the consensus, moreover it creates the Appia channel and push a consensus request event down to the next layer of the stack.

Next layer, Consensus layer processes the incoming event from upper layer, proposes a random binary value and wait for the messages in phase 1. Then it broadcast that proposal via Best Effort Reliable broadcast. Upon the delivery of messages from below layer, now at phase 1 it collects the messages and process for specific conditions and once majority replies received moves to phase 2, broadcast the phase 1 proposal and wait for phase 2 replies. Upon delivery of phase 2 messages it again checks the specific conditions to decide or re-propose a value moving to a new round. If decided, the decision will be

delivered to the upper application layer and at the same time decision will broadcast using Eager reliable broadcast to notify the other processes.

Below figure 2 depicts a scenario which needs to be handled as an additional feature of the algorithm.

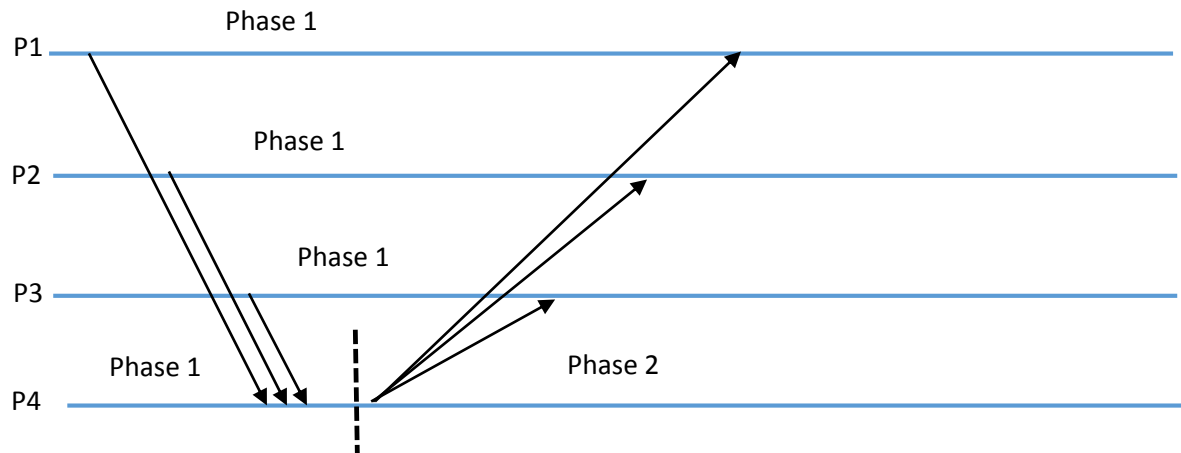


Figure 2: P4 moves to phase 2 while others still in Phase 1 but receive Phase 2 messages.

Here in this scenario there is a possibility for a certain process to be in phase 1 but receives phase 2 messages from another process. If we discard the messages which are not in the same phase we will lose the messages for the processes to move forward in the consensus phases. Therefore as a solution we buffer all the messages we receive outside from the current phase in order to use later.

Moreover we included the notion of consensus instance which is an incrementing value so that once a decision is made in a certain consensus instance we can safely discard the messages from previous consensus instances. This also ensures that if a process has decided a value in a much recent consensus instance, it will not decide a value with an older consensus instance which could arrive much later due to a network delay.

We extended the randomized binary consensus to randomized consensus for larger domain with some modification to the above implementation. Every process selects a random value from a domain it knows and disseminates it as an initial proposal. Every process collects the received proposals in an array. A process then initializes the common coin with domain values ensuring that the coin always output a value that has been proposed. To sample random elements with uniform distribution our implementation eliminates duplicates on the domain values received from phase 1.

In order to verify the completeness of our algorithm we implemented a simple java grid user interface, so that for each process owns a cell in a certain row. For binary consensus it moves up or down according to the decision. We extended the display mechanism with colors for large domain. When application layer receives a decision from the consensus layer, it sends a message to this demonstration UI via a socket connection.

## Section 3: Evaluation

We have provided a complete package of implementation as a jar file with all the supporting configuration in a folder named “DEPLOYMENT”. Execute the bat file to simple start the processes or else the commands to start the Demonstration application and the consensus application can be found in the bat file itself.

Once all the processes are started, instructions are displayed in console output. User only needs to trigger one process, it will trigger other processes to propose values automatically. Below figure 3 is an instance of our demonstration UI which illustrates the correctness of the algorithm.

Color Scheme [Domain = {1, 2, 3, 4}]

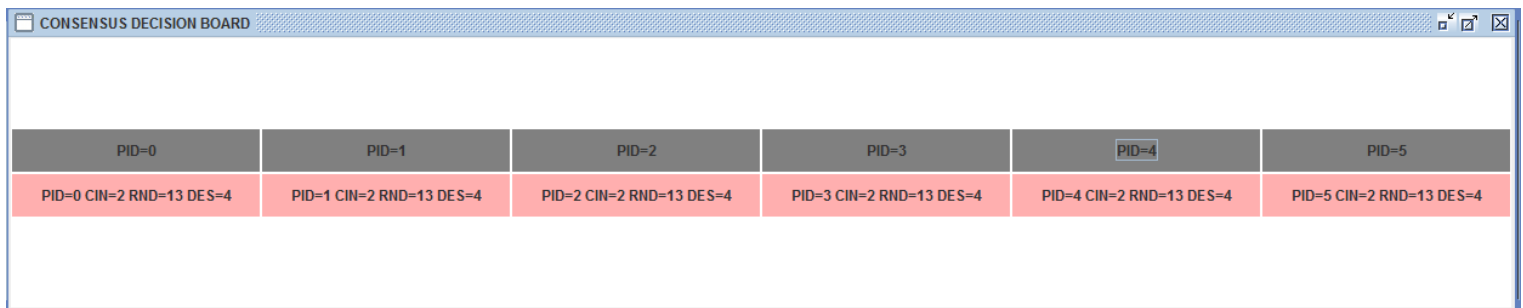
DES=1 → Move UP color GREEN

DES=2 → Move UP color BLUE

DES=3 → Move DOWN color YELLOW

DES=4 → Move DOWN color PINK

GRAY color shows the previous position of the process. If a cell appear in RED, it means that process has been crashed.



PID=0	PID=1	PID=2	PID=3	PID=4	PID=5
PID=0 CIN=2 RND=13 DES=4	PID=1 CIN=2 RND=13 DES=4	PID=2 CIN=2 RND=13 DES=4	PID=3 CIN=2 RND=13 DES=4	PID=4 CIN=2 RND=13 DES=4	PID=5 CIN=2 RND=13 DES=4

Figure 3: All processes deliver same value at consensus instance 2. 13 rounds has been spent in order to decide the value 4.

Moreover, we use the random value selection from a known domain at phase 2 if all processes send the special value ( $\perp$ ). This random selection ensures the algorithm’s termination property.

## Section 4: Conclusion

We implemented, tested and verified the Randomized Consensus. As a guideline we followed the course text book but we came across that even though it explains the basic intuition of the algorithm lot of small modifications and extensions to the algorithm such as message buffering is required at the actual usage. Furthermore, we got a better idea regarding the reliable broadcast because it is also used as a supportive tool to solve consensus. We believe that this project was really useful to apply the theoretical concepts learned in practice.