

# Virtual Synchrony - Project Report

Distributed Systems 1, 2017 - 2018

Daniele Bissoli - 197810, Andrea Zampieri - 198762

## 1 Introduction

In this project, we have implemented a simple group communication service that permits different members of a group to communicate through messages exchange. The management of the group is performed in a centralized fashion through a special node, named *Group Manager*. As design requirement, the system provide is able to guarantee the Virtual Synchrony, providing the synchronization between members.

### 1.1 View and Virtual Synchrony

A view is a local representation of the system, that is the set of group members that are still operational [**? maybe someone has crashed in the view, this sentence maybe false**]. Each member of the group has its own view and they have to globally agree on the same view in order to consistently communicate. Moreover, a view can be seen as a system epoch, since it changes every time a new participant join the group or when a cohort is detected to be crashed. Under the assumptions of reliable links and FIFO channels, *Virtual Synchrony* is the mechanism which makes all group members synchronize upon the same view. In this way messages are guaranteed to be delivered to all the currently operational nodes. [**? VS doesn't synch (in the sense that all the nodes wait for the others to catch up like in a "concurrent" approach), instead VS assures that messages that are sent in an epoch are delivered to all and just the nodes in that epoch**]

## 2 Project Architecture

The project is divided into three main classes (*Node*, *Participant* and *Group Manager*). In addition, it contains the definition of all exchanged messages and the class in charge to manage the creation of the system. In the following paragraphs the three main classes will be explained.

### 2.1 Node

This is the main class, which represent a common group member (also called *participant*). It defines the main functionalities of a group member which communicate under the Virtual Synchrony. These are:

- **reception[? maybe receipt] of a message** - when a data message is received, it is delivered to the application, but a copy of it is kept until the stability of it is reached, which means that the sender was able to send that message to everybody in the group. [**ALT: ...but a copy of it is kept until it becomes "stable"**]

- **stability of a received message** - upon the reception [? **receipt?**] of this message, the corresponding [? **copy of the**] data message can be deleted since it is confirmed that the [? **it**] message has been delivered to everybody in the group.
- **management of views** - when a view change message is received, it is responsible to check for unstable messages. If so, send them to all the group participants and then send a flush message that signal its intention changing the view
- **view change mechanism** - once a flush message is received, it checks if all the flush for that view has been received. If so, it performs a view installation (progress to next view). If there were previous views that were still open, it first install them [? **, deliver all messages not delivered for that particular**] and then it install the latest [? **possible**] one.

## 2.2 Participant

It represent a member of the group which continuously sends and receives message from others. It extends the *Node* class providing more functionalities such as send of data messages and mechanisms to set the participant as crashed. In addition, it provides a finer implementation of some functionalities as flush messages management and view change.

It inherits all the attributes of *Node* (e.g. the list of received messages, the unstable ones (with respect to each view) and the queue for messages received too early), and adds some of its own in order to be able to control the flow and the behaviour of the actors. Other than trivial ones (messageID, MIN\_DELAY, MAX\_DELAY for handling the transmission of new messages) there are some status variables to enforce the wanted behaviour.

- ***justEntered***: it's used just once to trigger a new *SendDatamessage* that makes the process send a new multicast, and reissues itself with a random delay  $d \in [\text{MIN\_DELAY}, \text{MAX\_DELAY}]$
- ***allowSending***: as its name tells, when its value is **false**, the process won't send any new message to the others in the system
- ***crashed***: tells whether the process has failed or not (it's used to simulate the crash of a given process); this variable can be set in different manners

As described in the Virtual Synchrony protocol, once a node receives a message informing of a view change, all the unstable messages for the previous epoch are sent to all the participants in the new view, ensuring that all the operational processes receive all the messages sent from everybody within its epoch. This avoids the situation in which some node don't receive certain messages from a crashed node, catching up thanks to the non-faulty ones still running. This kind of message is differentiated from the *standard* one, since when this kind of messages is received, no copy of it occurs since it can be seen as *already stable*.

Stuff to discuss: - explain how flush management works (e.g receivedFlush.isEmpty())  
 - slightly different protocol from the one given by Timofei - how timeouts after flush timeouts are avoided

## 2.3 Group Manager

It is a *reliable* group participant responsible to coordinate view updates between group members. It is built on top of the *Node* class and therefore it share most of the char-

acteristics of a group participant. Indeed, it also implement Virtual Synchrony logics, such as reaction to View Change and flush messages, although it is only able to receive messages. Besides that, it is in charge of detecting crashes, which is recognized through timeout mechanism based on:

- **missing heartbeat message** - it is a message sent continuously by each cohort to the Group Manager to notify that it is still operational. Group manager detects that a participant has crashed when its heartbeat is not received within a predefined timeout
- **missing flush message** - it is a message sent by all the group members when a view change is triggered. Group Manager detects if a participant is crashed when it does not receive its flush message within the expected predefined timeout

In both cases, the group manager will trigger a view change, creating a new view and sending it to every group member through multicast exchange.

Apart from previous tasks, Group manager also manage requests from new participants to join the group. To respect the Virtual Synchrony, a new view is generated to keep track of new member acquisition.

## 3 Implementation Details

### 3.1 Management of flushes and view-changes

The implementation in our project is slightly different from the one later proposed by Timofei Istomin. The following pseudocode and the comment should clarify the process.

---

**Require:** flush message  $f_a(v_i)$  from node  $a$  for new view  $v_i$   
*TODOFINISH*

---

When a flush message is received, the process checks whether all the flushes have been collected from the other participants for the installation of a successive view  $v_j$ . If so, all the successive (and pending) views to the current  $v_i$  are installed in order. This can be the case since faulty nodes may not send the flush message and leave some views pending, thus these ones need to be cascadingly installed once a more recent one meets the requisites.

Each view  $v_k$  s.t.  $i < k \leq j$  gets installed and delivers the messages that may be queued for the right view to deliver.

The method returns a boolean value that's consumed by the participants (the ones that send messages) in order to set their variable `allow_sending`

## 4 Crashes and Joining

The system is able to tolerate silent cohort crashes by means of the flush protocol. To control cohorts state, it is possible to send them special messages that make them crash according to predefined behaviour. Crashing modalities are specified below:

- **crash at any time** - immediately set the chosen node as crashed

- **crash on sending** - set the node crashed when it will perform next multicast to the group. As a result, only half of the group members will receive the sent message. It is up to the *all-to-all* exchange, performed during view change, to make all the other operational nodes delivering the sent message.
- **crash on receiving** - set the node crashed when it will receive next message. In this way the node is not able to perform the delivery of received message before crashing.
- **crash on view change** - set the node crashed when next view change is triggered. Therefore, the chosen node will not be able to perform the all-to-all message exchange and to send flush messages.

The system also allows to add further members. To join a new participant, a request must be sent to the group manager, which will to fulfill it. The Group Manager will generate a new *id* for the new member and it will introduce the participant into the group, triggering a view change.