Time Stamped Anti-Entropy protocol

Authors: Joan Manuel Marquès, Antonio González, David Mor, Montse Rovira

Distributed Systems course

Spring 2021

Assignment Outline

The aim of this practical assignment is to implement and evaluate a weak-consistency protocol for data dissemination.

The project consist on:

- Implementing the Time Stamped Anti-Entropy (TSAE) protocol [1] into an application that stores cooking recipes in a set of replicated servers.
- Add a remove operation on the recipes application
- Evaluate how TSAE behaves under different conditions.

Time Stamped Anti-Entropy (TSAE) protocol

Time Stamped Anti-Entropy (TSAE) [1] protocol is a weak-consistency protocol that provides reliable and eventual delivery of issued operations.

To get an overview and main ideas about TSAE read from [2]:

1. Section *1.Introduction* (excluding subsection 1.1)
2. Section *2.2 Timestamped anti-entropy* (also interesting to read: *2.1 Kinds of consistency*)

Then, to get all details about the protocol, read *Chapter 5. Weak-consistency communication* from [1]. More precisely, read:

- *5.1.1 Data structures for timestamped anti-entropy*
- *5.1.2 The timestamped anti-entropy protocol*
- (If you implement purge) *5.3 Purging the message log*. Instead of using vector acks (loosely-synchronized clocks) as described in this section, you should use unsynchronized clocks, described in section *5.4.4 Anti-entropy with unsynchronized clocks*.

Groups

Phase 1: should be done **individually**.

You are strongly advised to do the **phases 2 to 4 in groups of 2 students**, even though it is also possible to do it individually.

To Deliver

Two deliverables (more details in each phase):

1. phase 1 (theoretical exercises and practice)
2. phases 2 to 4 or second theoretical exercise

Deadlines are in the course schedule.

D1. First Deliverable

The work of this delivery must be done **individually**.

1. Phase 1: Theoretical exercise of TSAE protocol

1.1 TSAE protocol exercise (no purged log)

Exercise 1

Let's suppose that there are 3 hosts (A, B, C) that use TSAE protocol to exchange operations. At the initial time (t0) all hosts have the same state and their logs and summary vectors are as follow:

Summary A= A3, B3, C2	Log A= A1, A2, A3, B1, B2, B3, C1, C2
Summary B= A1, B4, C2	Log B= A1, B1, B2, B3, B4, C1, C2
Summary C= A2, B3, C4	Log C= A1, A2, B1, B2, B3, C1, C2, C3, C4

A1 stands for first operation from host A, B2 stands for the second operation from host B, and so on.

Let's assume that each anti-entropy session starts and ends at the same instant.

1.1.1 For the following sequence:

Time	Operation
1	Host A executes operation A4
2	Host B does an anti-entropy session with host C
3	Host C executes operations C5
4	Host A does an anti-entropy session with host C
5	Host A executes operation A5
6	Host B executes operations B5
7	Host A does an anti-entropy session with host B
8	Host C executes operations C6
9	Host B executes operation B6, B7, B8
10	Host C executes operations C7, C8
11	Host A executes operations A6
12	Host B does an anti-entropy session with host C
13	Host C does an anti-entropy session with host A
14	Host A executes operation A7

We want you to provide us with:

1. The data structures and operations exchanged during the anti-entropy sessions
2. The data structures (log and summary) at each host after each anti-entropy session
3. Indicate if the final state is consistent, i.e. all hosts have received the same operations. In case it

is not consistent, indicate which sessions should be done to reach a consistent state.

(Note: log is not purged)

1.1.2 For the following sequence:

Time	Operation
1	Host A executes operation A4
2	Host A does an anti-entropy session with host C
3	Host B executes operation B5, B6
4	Host A does an anti-entropy session with host B
5	Host C executes operation C5
6	Host A executes operation A5
7	Host C does an anti-entropy session with host A
8	Host A does an anti-entropy session with host B

We want you to provide us with:

1. The data structures and operations exchanged during the anti-entropy sessions
2. The data structures (log and summary) at each host after each anti-entropy session
3. Indicate if the final state is consistent, i.e. all hosts have received the same operations. In case it is not consistent, indicate which sessions should be done to reach a consistent state.

(Note: log is not purged)

LSimLogger.log

1.2 TSAE protocol exercise (purged log)

Imagine a situation with 5 hosts with unsynchronized clocks and log purging where Hosts A and B have the following AckSummaries:

AckSummary A

A	A 5	B 2	C 1	D 2	E 3
B	A 4	B 3	C 4	D 2	E 4
C	A 2	B 2	C 5	D 2	E 2
D	A 3	B 3	C 4	D 3	E 4
E	A 4	B 3	C 1	D 1	E 4

AckSummary B

A	A 4	B 2	C 3	D 4	E 5
B	A 2	B 4	C 3	D 2	E 5
C	A 3	B 2	C 3	D 3	E 5
D	A 3	B 1	C 3	D 4	E 5
E	A 4	B 1	C 3	D 4	E 5

a) Which is the content of Log in host A?

b) Which is the content of Log in host B?

A and B do an anti-entropy session. During the session both know who each other is (this is different from the algorithm in the Golding thesis).

c) Which operations are exchanged during the anti-entropy session?

d) Which AckSummary and log have each host after ending the session?

1.3 TSAE protocol exercise

Question 1: In TSAE, why is it needed to purge the log periodically? Explain with your own words the question.

Question 2: Describe what are the main fields of a TSAE data structure.

2. Phase 1: Implementation and testing of Log and TimestampVector data structures

Phase 1 will consist of implementing the methods from `Log` and `TimestampVector` data structures.

In this phase only *add* operations are issued. Don't implement the functionality to purge the log.

Annex A includes details about the timestamps used in this practical assignment.

NOTE: be **careful with concurrent access to data structures**. Two actions issued by different threads may interleave. **Use** some **synchronization mechanism to avoid interference**.

Implement `Log` and `TimestampVector` data structures according to the TSAE protocol described in section *Time Stamped Anti-Entropy (TSAE) protocol*.

Environment

Requires Java 7.

We recommend you to use eclipse as IDE. We will provide you an Eclipse project that contains the implementation of the cooking recipes application except the parts related to TSAE protocol.

All scripts for running local tests are prepared for Ubuntu-linux but other OS can be used. In that case, you will be responsible for adapting the scripts to your OS.

2.1. Test locally

To test locally `Log` and `TimestampVector` data structures run:

```
$  java -cp ../bin:../lib/* recipes_service.Server --phase1
```

(run it from `scripts` folder)

Introduce a recipe and check if `Log` and `TimestampVector` data structures are correct.

```
$  ./start.sh 20004 --phase1
```

(run it from `scripts` folder)

$1: listening port of `Phase1TestServer`, the server that tests the correctness of your solution. In case that port `20004` is already in use by another application you can change it to any other unused port.

Executes `Log` and `TimestampVector` with a predefined set of users and operations and compares it with a `Log` and `TimestampVector` previously calculated.

2.2. Formal evaluation of phase 1

It is compulsory to **use DSLab to assess the phase 1** of the practical assignment. Therefore, once your application runs in local, upload `Log` and `TimestampVector` classes into DSLab (http://sd.uoc.edu/dslab) and assess them:

1. Create a *Project* and upload `Log` and `TimestampVector` classes into this project.
2. Create an *Experiment* that executes the project created in step 1.
3. Check the result of the execution of step 2.

2.3. Things to deliver

A **file** according to the following convention:

Deliverable1-Year-FamilyName1_Name1.zip

This file should **include**:

- solution of the theoretical exercise.
- phase1.pdf: a (short) report detailing and explaining all decisions taken. A proposal of report template is included in the practical assignment distribution (*/doc* folder).

  In addition, you should detail the portions of your source code you consider are most significant.
- Source code: `Log` and `TimestampVector` classes.

The zip file should have a single directory named like the zip file with the following structure (use same structure and names):

Subdirectory	Content
`/doc`	Solution theoretical exercise Report in pdf
`/src`	`Log` and `TimestampVector` classes.

D2. Second Deliverable

This second deliverable has two options:

A. Implementation of phases 2 to 4
B. Theoretical exercises

Check the *Evaluation* section for details regarding grading. Main aspect are:

- Option A allows students to obtain a grade up to an A.
- Option B allows students to obtain a grade up to a C+.
- Students that implement phases 2 to 4 (option A) are not required to do the theoretical exercise to be able to get the highest grade.

Option A. Implementation of phases 2 to 4

Phases 2 to 4 might be done individually or in groups of two students.

Groups will be created when the period of the second deliverable of the practical assignment starts.

A.1. Phase 2: Implementation of a reduced version of the application and TSAE protocol: only add operation; no purge of log

Implement the TSAE protocol described in section *5.1.2 The timestamped anti-entropy protocol* (from [1]) in the following classes (Package: recipes_service.tsae.sessions):

- TSAESessionOriginatorSide: Originator protocol for TSAE (figure 5.7 without acks)
- TSAESessionPartnerSide: Partner's protocol for TSAE (figure 5.8 without acks)

ServerData class (package recipes_service) contains Server's data structures required by the TSAE protocol (log, summary, ack) and the application (recipes).

- You can add any required method to allow TSAESessionOriginatorSide and TSAESessionPartnerSide manipulate these data structures.

Use the following methods to send and receive data between servers (package communication):

- readObject() from ObjectInputStream_DS class.
- writeObject() from ObjectOutputStream_DS class.

Use the following message classes for the communication between partners:

(package recipes_service.communication)

- MessageAErequest: message sent to request an anti-entropy session.
- MessageOperation: message sent each time an operation is exchanged during an anti-entropy session.
- MessageEndTSAE: message sent to finish an anti-entropy session.

In this phase:

1. Only *add* operations (addRecipe method in ServerData class) are issued.
2. No purge of Log.

NOTE: be **careful with concurrent access to data structures**. Two actions issued by different threads may interleave. **Use** some **synchronization mechanism to avoid interference**.

Adaption of pseudocode of figures 5.7 and 5.8

As part of the evaluation of your implementation of the TSAE protocol we will run your solution interacting with the teachers' solution. To make sure that both implementations agree the implementation of TSAESessionOriginatorSide and TSAESessionPartnerSide classes should follow the following templates:

TSAESessionOriginatorSide

```
// Send to partner: local's summary and ack
      ...
      Message     msg = new MessageAErequest(localSummary, localAck);
      out.writeObject(msg);

      // receive operations from partner
      msg = (Message) in.readObject();
      while (msg.type() == MsgType.OPERATION){
            ...
            msg = (Message) in.readObject();
      }

      // receive partner's summary and ack
      if (msg.type() == MsgType.AE_REQUEST){
            ...
            // send operations
            ...

            // send and "end of TSAE session" message
            msg = new MessageEndTSAE();
            out.writeObject(msg);

            // receive message to inform about the ending of the TSAE session
            msg = (Message) in.readObject();
            if (msg.type() == MsgType.END_TSAE){
                  //
            }
      }
```

```
TSAESessionPartnerSide
// receive originator's summary and ack
      msg = (Message) in.readObject();
      if (msg.type() == MsgType.AE_REQUEST){
            ...

            // send operations
            ...
            out.writeObject(msg);

            // send to originator: local's summary and ack
            ...
            msg = new MessageAErequest(localSummary, localAck);
            out.writeObject(msg);

            // receive operations
            msg = (Message) in.readObject();
            while (msg.type() == MsgType.OPERATION){
                  ...
                  msg = (Message) in.readObject();
            }

            // receive message to inform about the ending of the TSAE session
            if (msg.type() == MsgType.END_TSAE){
                  // send and "end of TSAE session" message
                  msg = new MessageEndTSAE();
                  out.writeObject(msg);
                  }
      }
```

Your Tasks in Phase 2

Implement the protocol (without acks).

To test if your solution works properly use the provided test environment. Section 4 contains more details about it.

Run phase 2 Locally

To test your implementation locally in your computer use the shell script start.sh (scripts folder).

This script has many parameters. Some useful examples:

(arguments are explained only on its first appearance)

```
$  ./start.sh 20004 3 --menu --nopurge
```

Runs 3 Servers and a TestServer locally.

$1: listening port for the TestServer, the server that tests the correctness of your solution. In case that port 20004 is already in use by another application you can change it to any other unused port.

$2: number of Servers to be instantiated

--menu: runs in menu mode. (without --menu argument, it will run in simulated mode, i.e. user activity (add recipes) and dynamism (connections and disconnections) will be automatically generated.

--nopurge: Log is not purged.

```
$  ./start.sh 20004 15 --logResults --nopurge --noremove
```

Runs 15 Servers and a `TestServer` locally.

User activity (add recipes) and dynamism (connections and disconnections) is automatically generated.

Results will be stored in a file named as the *groupId* from `config.properties` file (on current path).

`--logResults`: log results in the following two files:

- <*groupId*[2]>: log of all executions and the result for each of them.
- <*groupId*[3]>`.data`:
  - in the case that all solutions are equal, contains the final state of data structures.
  - in the case that NOT all solutions are equal, contains the first two found solutions that were different.

`--nopurge`: `Log` is not purged.

`--noremove`: no remove operations are issued.

```
$  ./start.sh 20004 15 --logResults -path ../results --nopurge --noremove
```

`-path <path>`: result files are created in `../results` folder. If no `-path` is specified (as in the previous example) files will be stored in the current folder.

We recommend you to **start** testing your application using the `--menu` option, which will allow you to test the TSAE protocol and data structures at your pace.

Then execute your implementation locally without the `--menu` parameter to test your application under similar conditions that the ones that will be used in the formal evaluation.

Finally, once your application runs in local, upload it in DSLab web for its formal evaluation. In DSLab, your implementation will run in a real distributed environment interacting with instances (also distributed) of an implementation done by the professors of this course.

(REMEMBER: use DSLab web for its formal evaluation)

Run phase 2 at DSLab (evaluation framework)

Similarly to phase 1, **use DSLab to assess the phase 2** of the practical assignment. The practical assignment is going to be transparently executed in a real distributed environment that includes instances of your implementation and of the instructor's implementation.
First of all, you must create a group. Then, you are able to upload your solution for the practical assignment, run it and check the result of all your executions.

(DSLab website: http://sd.uoc.edu/dslab)

Building projects

First you need to create a new project uploading the necessary Java classes (.java) from your local machine. For example, you can create a project for each phase.

2 *Name of the file will be the value of groupId property in config.properties file (scripts folder).*

3 *Value of groupId property in config.properties file (scripts folder) will be the first part of the file name. i.e. file name: <value of groupId property>.data.*

Classes to upload to DSLab in phase 2:

- Log.java
- TimestampVector.java
- TSAESessionOriginatorSide.java
- TSAESessionPartnerSide.java
- ServerData.java (package recipes_service)

Secondly, you are able to build the project in order to use it for the next step.

Launching experiments

A built project can have as many experiments associated as you need. Thus, selecting one built project, you can configure and launch an experiment into the remote platform transparently.

Checking results

Once the experiment execution is finished, you can review your results in your experiment information.

A.2. Phase 3: Extension of phase 2 to purge log with unsynchronized clocks

Extend previous phase adding all required logic to purge the log when using unsynchronized clocks:

- Implement purging method in Log class.
- Implement the necessary methods of TimestampMatrix according to section *5.4.4 Anti-entropy with unsynchronized clocks*.
- Extend the implementation of TSAE protocol of phase 2 to include ack exchange and to purge log (Section *5.3 Purging the message log*. Remember that you must use unsynchronized clocks explained at section *5.4.4 Anti-entropy with unsynchronized clocks*)

In this phase only *add* operations are issued.

Your tasks in Phase 3

Implement the extensions above described and test them in the test environment.

Run phase 3 Locally

To test phase 3 locally:

```
$ ./start.sh 20004 3 --logResults -path ../results --menu
```

Runs 3 Servers and a TestServer locally (and result files are created in ../results folder).

Log is purged.

Menu mode.

```
$ ./start.sh 20004 15 --logResults -path ../results --noremove
```

Runs 15 Servers and a TestServer locally (and result files are created in ../results folder).

Log is purged.

Simulated activity and dynamism mode.

No remove operations.

Run phase 3 at DSLab

(REMEMBER: use DSLab web for its formal evaluation)

Classes to upload to DSLab in phase 3:

- Log.java
- TimestampVector.java
- TimestampMatrix.java
- TSAESessionOriginatorSide.java
- TSAESessionPartnerSide.java
- ServerData.java

A.3 Phase 4: Evaluation of TSAE protocol and implementation of Remove recipe operation

Phase 4.1 Extend application adding the remove recipe operation

Extend the cooking recipes application implementing the *remove recipe* operation:

1. Identify the problematics that introduce the *remove recipe* operation. Illustrate it with an example.
2. Design and implement a proposal to remove recipes and test it in the running environment.

Modifications should be done in the following methods of `ServerData` class (package `recipesService`):

- `removeRecipe`
- any other required method added while implementing the TSAE protocol in phase 2.

Phase 4.2 Evaluation of TSAE protocol

Run the practical assignment:

- in different environments:
  - local: all instances of the service in a single computer
  - distributed (realistic environment): instances of the service running in different computers distributed on the Internet (and, therefore, connected by a real network).
- under different conditions:
  - scale: number of servers
  - dynamicity: different degrees of connection and disconnection
  - accelerate propagation: periodicity of anti-entropy sessions, to start disseminating new data right after the data is generated, number of sessions with different partners that are done each time an anti-entropy session is scheduled.
  - level of activity generation (add and remove operations)

Evaluate the impact of parameters on the behavior of TSAE. Also compare local and distributed settings. You should detail the experiments done and the obtained conclusions. Current implementation includes a basic modeling of parameters. You are encouraged to improve this modeling to get a more realistic one. Changes must be justified.

Your tasks in Phase 4

Implement the *remove* recipe *operation* and test it in the testing environment.

Do a rapport describing how parameters and running environment influence on TSAE protocol performance.

Run phase 4

Use the script *start.sh* explained in previous phases.

In *phase 4.1* (extend application adding remove recipe operation), run *start.sh* script to test your implementation locally. For its formal evaluation use DSLab web.

```
$  ./start.sh 20004 15 --logResults -path ../results
```

Runs 15 Servers and a TestServer locally (and result files are created in ../results folder).

Log is purged.

Simulated activity and dynamism mode.

Remove operations are issued.

(REMEMBER: use DSLab web for its formal evaluation)

Classes to upload to DSLab in phase 4.1:

- Log.java
- TimestampVector.java
- TimestampMatrix.java
- TSAESessionOriginatorSide.java
- TSAESessionPartnerSide.java
- ServerData.java

In *phase 4.2* (evaluation of TSAE protocol), modify parameters of config.properties file (scripts folder) to evaluate TSAE protocol under different conditions and environments.

- Use *runN.sh* script to run N times *start.sh* script. Example:

  ```
  $  ./runN.sh 10 20004 15 --logResults -path ../results
  ```

  runs 10 times *start.sh* script with the following parameters:

  ```
  $  ./start.sh 20004 15 --logResults -path ../results
  ```

- Modify activitySimulation class only in case that in phase 4 you want to better understand or modify the modeling of activity and dynamicity (package recipes_service.activitysimulation).

Phase 4.2 is not evaluated using DSLab. It consists of doing a rapport describing how parameters and other conditions influence on TSAE performance. Therefore, run the TSAE protocol with different values for parameter and under different conditions.

A.4. Things to deliver

A **file** according to the following convention:

Deliverable2-Year-groupId-FamilyName1_Name1-FamilyName2_Name2.zip

This file should **include**:

- A (short) report detailing and explaining all decisions taken. A proposal of report template is included in the practical assignment distribution (*/doc* folder).

  In addition, you should detail the portions of your source code you consider are most important, an explanation about how it works and the tests you used to validate your assignment.

  Finally, scripts and a detailed example of how to run your practical assignment.
- Source code: your eclipse project and the portion of source code you have implemented.
- If evaluation of TSAE protocol of phase 4.2 was done: a document explaining simulations done and results obtained. Name the file phase4.pdf.

The zip file should have a single directory named like the zip file with the following structure (use same structure and names):

Subdirectory	**Content**
`/doc`	Report in pdf Document phase4.pdf (in case of phase 4.2)
`/src`	Source code. Eclipse project with your implementation.
`/bin`	All required files to execute your practical assignment. Explained carefully in the report how to execute it.

Option B. Theoretical exercise

The theoretical exercise must be done **individually**.

See annex E for openssl commands.

Assignment outline

The aim of this assignment is to understand the main concepts of the block chain systems.

Block chain systems combine many concepts in order to give a solution to a distributed ledger with security.

In order to get the basic details of the Block Chain read the following article:

https://www.igvita.com/2014/05/05/minimum-viable-block-chain

After reading the full article the assignment consist on:

- Responding the "Theoretical questions"
- Resolve the "Practical exercises"

Theoretical questions

1. Explain blockchain in your own words and describe one of its uses other than cryptocurrencies.
2. Describe the main distributed consensus problems in a P2P network. Taking into account that centralization could solve them, what advantages and disadvantages does it have?
3. How is it possible that a double-spend transaction is generated? How would you avoid it in a large network?
4. When can we consider that a block is final?
5. What is the main drawback of a transaction confirmation without fees? How would you solve it?
6. Why is proof-of-work considered a requirement? What characterizes the proof-of-work solution in the minimum viable block chain?

Practical exercises

1. Sketch a network diagram (you can use Dia: https://wiki.gnome.org/action/show/Apps/Dia, or the alternative that better suits you) highlighting the main actors of the described block chain. (*Delivery should be in PDF format, this is not a UML diagram*).
2. Let's say you are a party interested in collecting fees by validating and confirming transactions. You have received enough transaction notifications (File*: transaction_notifications.csv headers in Annex D*) to compensate your proof-of-work costs so you are willing to generate as many blocks as possible. Now you have to validate all received transactions (remember to check for double-spends) and aggregate them into blocks of five transactions each.

   NOTE: The balance of each participant at t0 is:

   - A: 100€
   - B: 50€

   (suppose that all signatures verify correctly).

   a) Have you received double-spends? in this case indicate which transactions are not valid.

   b) Show which transactions get aggregated into which blocks.

   c) Which fields are required by a *ledger.csv* to support both transactions and block annotations?

   d) Write a ledger.csv file with the annotations for your blocks and the transactions that this blocks aggregate. (suppose fees for 0.1% for each transaction in a block)

   e) Which is the balance of D at the end of the exercise taking into account the 0.1% fees? (3 decimals)

3. Using the same *transaction_notifications.csv* file, plus the **.signature* files:

   NOTE: signatures made with the command:

```
echo 'A,B,€,1,45'|openssl dgst -sha256 -sign pki/A.key -out notification_1.signature
```

   a) Verify all signatures and indicate which signatures are correct and which don't verify. Remove invalid notifications, if any.

   b) Generate your key-pair and sign your block transactions from the previous exercise.

To deliver

A zip file named with your campus username in the form

campus_username-blockchain.sd.zip

with the structure:

Files	Description
`username.pdf`	Your answers to both theoretical and practical questions.
`student.crt`	Your public key.
`ledger.csv`	The ledger.csv from practical exercise 2.
`ledger-XX.signature`	Specifying the id of each signed block in the file name. This is, one signature file per signature.
`diagram_file.[png|jpg|pdf]`	The diagram from practical exercise 1.

B.1. Evaluation

Theoretical questions correct: C-

Theoretical questions + Practical exercise, both correct: C+

Annex A. Source code and documentation

Papers folder contains referenced paper and thesis that explain the TSAE protocol.

TSAE folder contains all packages and classes required for the practical assignment.

Important: **do not implement new classes**. **Modify only the classes indicated in each phase**.

(except if you modify the basic modeling of parameters in phase 4)

Classes that you should modify:

1. package recipes_service.tsae.datastructures:
   - Log: class that logs operations.
   - TimestampVector: class to maintain the summary.
   - TimestampMatrix: class to maintain the acknowledgment matrix of timestamps
2. package recipes_service.tsae.sessions:
   - TSAESessionOriginatorSide: Originator protocol for TSAE.
   - TSAESessionPartnerSide: Partner's protocol for TSAE.
3. package recipes_service:
   - ServerData: contains Server's data structures required by the TSAE protocol (log, summary, ack) and the application (recipes). You can add any required method to allow TSAESessionOriginatorSide and TSAESessionPartnerSide manipulate these data structures.
     - addRecipe method: adds a new recipe.
     - removeRecipe method: removes a recipe.

Classes that you should use but NOT modify:

4. package recipes_service.tsae.data_structures:
   - Timestamp: a timestamp. A timestamp allows the ordering of operations issued from the same host. It is a tuple <hostId, sequenceNumber>. Sequence number is a number that grows monotonically. The first valid timestamp issued by a host will have an initial value of 0. A negative sequence number means that the host hasn't issued yet any operation. Timestamps can not be used to order operations issued in different hosts. Next timestamp is obtained calling the method nextTimestamp() from class ServerData (package recipes_service).
5. package recipes_service.data:
   - AddOperation: an add operation (operations are logged in the Log and exchanged with other partners).
   - RemoveOperation: a remove operation (operations are logged in the Log and exchanged with other partners).
   - Recipe: a recipe.
   - Recipes: class that contains all recipes.

6. package recipes_service.communication)
   - MessageAErequest: message sent to request an anti-entropy session.
   - MessageOperation: message sent for each operation exchanged during an anti-entropy session.
   - MessageEndTSAE: message sent to finish an anti-entropy session.
7. package communication:
   - ObjectInputStream_DS: class that implements a modification of the ObjectInputStream to simulate failures. Use the method readObject().
   - ObjectOutputStream_DS: class that implements a modification of the ObjectOutputStream to simulate failures. Use the method writeObject().
8. package recipes_service.activitysimulation: Only in case that in phase 4 you want to better understand or modify the modeling of activity and dynamicity.

Annex B. Activity simulation and dynamicity

Simulation of communication failures

ActivitySimulation class (package recipes_service.activitysimulation) decides when to simulate the disconnection (or network failure) of a host and when to reconnect it.

To simulate communication failures, a disconnected host abruptly closes all its Input and Output streams, which results in an exception in the two partners that are using the stream.

Annex C. Details about the execution

Executions has two phases:

1st phase: during this phase nodes connect and disconnect, activity is generated (add and remove operations) and occur TSAE sessions. Field SimulationStop in config.properties file indicates the duration of this phase.

2nd phase: only TSAE sessions. No new activity is generated. Connected nodes won't disconnect. Disconnected nodes won't reconnect. Field executionStop in config.properties file indicates the duration of this phase. During this phase, each samplingTime (in config.properties file) seconds the node will send its state to be evaluated. Each time it sends its state is called an iteration. The number of iterations required to get a consistent state will indicate the degree of divergence among replicas when phase 1 ended.

Annex D. Headers

ID: The id for this annotation.

SHA-256 checksum: The checksum for this annotation.

From: The code for the sender, if any.

To: The code for the receiver, if any.

What: The origin/s from which the amount for this annotation is obtained.

Time: The moment when this transaction happened.

Value: The amount exchanged in this annotation, if any.

Annex E. PKI reference

Generate your private and public keys:

```
openssl req -nodes -x509 -sha256 -newkey rsa:4096 -keyout my.priv.key -out my.crt
```

Sign / Verify:

Sign with private key:

```
openssl dgst -sha256 -sign my.priv.key -out transaction.signature transaction.csv


echo 'Real important text'|openssl dgst -sha256 -sign my.priv.key -out important.signature
```

To hash the previous signature:

```
openssl dgst -sha256 transaction.signature
```

To verify with public key, the transaction previously signed:

```
openssl dgst -sha256 -verify <(openssl x509 -in my.crt -pubkey -noout) -signature transaction.signature transaction.csv
```

Asymmetric encryption

Encrypt w/ public key:

```
openssl rsautl -encrypt -inkey <(openssl x509 -in my.crt -pubkey -noout) -pubin -in plain.txt -out cyphered.txt.enc
```

Decrypt w/ private key:

```
openssl rsautl -decrypt -inkey a_blockchain.key -in cyphered.txt.enc -out decyphered.txt
```

References

[1] **Richard A. Golding** (1992, December). Weak-consistency group communication and membership. Ph.D. thesis, published as technical report UCSC-CRL-92-52. Computer and Information Sciences Board, University of California. Santa Cruz. **(chapter 5)**

[2] **R. Golding; D. D. E. Long**. The performance of weak-consistency replication protocols, UCSC Technical Report UCSC-CRL-92-30, July 1992.