Highly Dependable Systems

Sistemas de Elevada Confiabilidade

2023-2024

HDS Serenity Ledger

Stage 2

Goals

This project aims to develop a simplified permissioned (closed membership) blockchain system with high dependability guarantees, called HDS Serenity (HDS^2). In the second stage, we will build upon the work developed during the first stage and refine the application semantics and the dependability guarantees of HDS^2.

The second stage has the following goals:

- Provide a cryptocurrency application that allows clients (represented by accounts in the system) to perform cryptocurrency transfers between them.
- Strengthen the Byzantine fault tolerance guarantees of the system, including the ability to tolerate Byzantine clients.
- Experimentally confirm the correctness, performance, and performance under attacks, of your implementation. In particular, a single malicious replica should not be able to slowdown the entire system.

We retain the simplifying assumptions from the first stage that the participants (clients and servers) are static and known beforehand, together with the associated public keys and IPs.

Cryptocurrency application

A client of the system can transfer funds from their account (whose private key they own) to another client account, provided they have the necessary balance. The set of all accounts and the associated balance should satisfy the following dependability and security guarantees:

- the balance of each account should be non-negative;
- the state of the accounts cannot be modified by unauthorized users;
- the system should guarantee the non-repudiation of all operations issued on an account.

For simplicity, all accounts start with a pre-defined positive balance.

The client API has the following specification:

- transfer(PublicKey source, PublicKey destination, int amount, …)
  Specification: send a given amount from account *source* to account *destination*, if the balance of the source allows it. If the server responds positively to this invocation, it must be guaranteed that the *source* has the authority to perform the transfer.

- check_balance(PublicKey key,…)
  Specification: obtain the balance of the account associated with the *key* passed as input.

Finally, all update transactions must pay a fee to the block producer (the leader). Students must reason about the implications of these requirements and explain in the report possible ways how the design could address these implications.

Behavior under Byzantine faults

Students must reason about the potential attacks that a Byzantine server or client can make, including collusion between clients and servers, and discuss and implement the appropriate countermeasures. As part of this second stage, students will demonstrate the ability of their system to withstand a variety of Byzantine attacks. In particular, the final report must include a section, entitled "Behavior under attack", that presents a systematic study of this aspect, by describing a set of experiments that you devised to show the correct behavior under attack. This study should be as thorough and systematic as possible, while obeying the page limits.

Implementation Steps

To help in the design and implementation task, we suggest that students break up the project into a series of steps, and thoroughly test each step before moving to the next one. Having an automated build and testing process (e.g.: JUnit) will help students progress faster. Here is a suggested sequence of steps:

- Step 1: Ensure that all the requirements from the first stage are satisfied. Extend the system to support Byzantine clients.
- Step 2: Define the syntax and semantics of application-level operations, implement the operations on the server and client side, and test the client and server interactions.

- Step 3: Implement a thorough set of tests that demonstrate that the implementation is robust against Byzantine behavior.

Submission

Submission will be done through Fénix. The submission must include:

- a self-contained zip archive containing the source code of the project and any additional libraries required for its compilation and execution. The archive shall also include a set of demo applications/tests that demonstrate the mechanisms integrated in the project to tackle security and dependability threats (e.g., detection of Byzantine behavior from blockchain members and clients). A README file explaining how to run the demos/tests is mandatory.
- a report of up to 4 pages addressing:
    - A short overview of the system design;
    - The most relevant implementation aspects;
    - The experimental evaluation that shows that the system is dependable in the presence of Byzantine behavior.
    - The report must use this template: https://www.usenix.org/conferences/author-resources/paper-templates

The deadline is **March 27 at 23:59**. More instructions on the submission will be posted in the course page.

Ethics

The work is intended to be done exclusively by the students in the group and the submitted work should be new and original. It is fine, and encouraged, to discuss ideas with colleagues – that is how good ideas and science happens - but looking and/or including code or report material from external sources (other groups, past editions of this course, other similar courses, code available on the Internet, ChatGPT, etc) is forbidden and considered fraud. We will strictly follow IST policies and any fraud suspicion will be promptly reported.