Highly Dependable Systems

Sistemas de Elevada Confiabilidade

2023-2024

HDS Serenity Ledger

Stage 1

Goals

This project aims to develop a simplified permissioned (closed membership) blockchain system with high dependability guarantees, called HDS Serenity (HDS^2). The system will be built iteratively throughout the first and second stages of the project.

At the consensus layer, we will use the Istanbul BFT Consensus Algorithm [1], or IBFT. This protocol, which is known in the blockchain community as QBFT, is now part of the Ethereum core specification [2] and implemented in several real-world blockchains [3].

The main goal of the first stage is to understand and then extend an initial and incomplete version of Istanbul BFT that is provided as a starting point [4]. In particular, this initial codebase does not handle faulty leaders (among other limitations) and your task is to extend it with the round change protocol to select a new leader. To simplify your code, the implementation can make the following two assumptions:

- The system membership is static for the entire system lifetime, including a pre-defined set of process identifiers {0, … , N-1}, which are used for choosing a rotating leader, starting with id 0. This membership information (namely the ids and network addresses of the system members that implement the blockchain, which we refer to as the blockchain members) is well-known to all participating processes before the start of the system (e.g., they can fetch this information from a static location).

- There is a Public Key Infrastructure in place. This means that blockchain members should use self-generated public/private keys, which are pre-distributed before the start of the system, and the public key information is also included in the static system membership.

Design requirements

The design of the HDS^2 system consists of two main parts: (1) a client library that is linked with the application that users employ to access the system, and (2) the server-side logic responsible for keeping the state of the system and collectively implementing the blockchain service. The library is a client of the blockchain, and its main responsibility is to translate application calls into requests to the service, namely to trigger instances of the consensus protocol. Any client messages that do not obey the expected protocol behavior must be detected by the blockchain service and handled appropriately. The servers run the IBFT Consensus Algorithm and maintain state of the system, as detailed below.

The following assumptions are done on the system's components and environment:

- The application can trust its local instance of the client library. This means that we are not concerned with attacks where a compromised library tries to attack an honest application.
- A subset of up to 1/3 of the blockchain members may be malicious and behave in an arbitrary manner.
- The network is unreliable: it can drop, delay, duplicate, or corrupt messages, and communication channels are not secured. To address this, the basic communication in the provided implementation is done using UDP, and then various layers of communication abstractions are implemented. Using UDP allows the network to approximate the behavior of Fair Loss Links, and then the initial implementation builds an (incomplete) subset of the abstractions we learned in class. Your task is to identify the limitations of the set of abstractions that are currently implemented, and, if necessary, add new layers of channel abstractions on top of those.

For this stage, we will use a simplified client that only needs to submit requests to the service (in stage 1, a request is a simply an invocation of <append, string> with the string to be appended to the blockchain) and then waits for a reply confirming if and where in the sequence of blocks the request was executed. The semantics of the interactions between clients and the service will be refined in the second stage of the project.

Note that there is a gap between the interface described in the previous paragraph (clients whose interface is a request over the form <append, string>, and which should be separate machines from the members of the blockchain) and the interface of the Istanbul BFT protocol (i.e., the START procedure and the DECIDE output described in Algorithms 1 and 2 of the paper, both running on the blockchain members). We leave it as an open design decision to devise a mechanism to fill this gap, i.e., translate client requests to invocations of the Istanbul BFT protocol, and also for the respective response path. When filling this gap, there should also exist an upcall that is invoked whenever the DECIDE output is reached, for notifying the upper layer of a state update. This upcall can be handled by a simple service implementation that keeps an array with all the appended strings in memory.

You must also include a set of tests for the correctness of the system. For these tests to be thorough, they need to be more intrusive than simple "black box" tests that submit a load consisting of strings to be appended to the blockchain: in particular, there must be a way to change the way that messages are delivered or the behavior of Byzantine nodes to test more challenging scenarios.

Implementation requirements

The provided implementation is written in Java, and it does not yet make use of cryptographic primitives. As such, your solution should use the Java Crypto API to implement the cryptographic functions.

We will value the following aspects of your implementation: (1) the correctness of the protocol implementation, (2) the quality of your code and associated report, (3) the performance of the implemented protocols, and (4) quality of the test suite.

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: As a preliminary step, before starting any implementation effort, make sure you have carefully read and understood Istanbul BFT (namely the view change component, which you must implement in this stage). Reason about the design and implementation challenges that might arise before moving to step 2.
- Step 2: Complete the channel implementation that is provided with the modules for the missing layers.
- Step 3: Develop a simple client library, enabling you to have clients that trigger the execution of instances of BFT consensus.
- Step 4: Implement the round change component of the Istanbul BFT protocol. Always try to tackle this in an incremental fashion, by identifying successive steps of the round change protocol, and exhaustively testing each step before moving to the next step.
- Step 5: Develop a set of tests that exercise the developed features and show that the implementation is robust to Byzantine processes.

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 must also include a set of demo applications/tests that demonstrate the mechanisms integrated in the project to tackle security and dependability threats (e.g., tolerating Byzantine behavior from blockchain members). A README file explaining how to run the demos/tests is mandatory.
- a concise report of up to 4,000 characters addressing:
    - explanation and justification of the design, including an explicit analysis of the possible threats and corresponding protection mechanisms.
    - explanation of the dependability guarantees provided by the system.

The deadline is **March 8 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.

References

[1] Henrique Moniz. The Istanbul BFT Consensus Algorithm.

https://arxiv.org/pdf/2002.03613.pdf

[2] QBFT Blockchain Consensus Protocol Specification v1.

https://entethalliance.github.io/client-spec/qbft_spec.html

[3] EEA Publishes QBFT Blockchain Consensus Protocol.

https://entethalliance.org/23-01-qbft-spec-version-1-released/

[4] Simple Istanbul BFT Consensus (base implementation)

https://fenix.tecnico.ulisboa.pt/downloadFile/3096843219124421/sec2324-project-base-master.zip