Homework 4

- Follow the instructions very carefully. Answers that do not conform to the instructions will not be given credit.
- Understand thoroughly all the code given to you in this lab. Search for documentation online if there is a primitive or API you have not encountered before.
- **Submission:** You may only modify the `UnauthBFTNode.java` and the `MajorityVotingMaliciousNode.java` classes. You may also submit your own helper classes that the grader will place under the `hw4.consensus.bft` package. No changes to any other project classes will be accepted as part of your submission.

In this project, you will implement the basic, unauthenticated Byzantine fault tolerant consensus algorithm described by Lamport, Shostak, and Pease in their seminal paper titled *The Byzantine General's Problem* [1].

The project code provides you with a simple consensus protocol testing framework that you will use to test the correctness of your implementation. Read and thoroughly understand all the project code before beginning.

The Java class named `hw4.BasicTests` gives several tests that demonstrate how to use the consensus testing framework. Your submission should make all the tests pass. For example, `test0` test case is reproduced below and configures a small network running the *FollowLeader* consensus protocol described in class that fails to solve the BFT consensus problem for any number of malicious nodes.

```
@Test
public void test0() {
    int NUM_ROUNDS = 5;
    int NUM_NODES = 4;

    Network net = new Network();
    net.newNodes(FollowLeaderNode.class, NUM_NODES);

    Node leader = net.getNodes().get(0);
    net.setLeader(leader)
            .setValueSet(VALUE_SET)
            .setDefaultValue(VALUE_0)
            .setRounds(NUM_ROUNDS);
    leader.setLeaderInitialValue(VALUE_1);

    NetworkUtil.completeGraph(net);

    SynchronousExecutor e = new SynchronousExecutor(net).run();
    assertTrue(e.isAgreementSatisfied());
    assertTrue(e.isValiditySatisfied());
}
```

The test case creates a `Network` object which represents a network of four FollowLeader nodes whose communication links form a complete graph. The first node is marked as the leader. The set of possible values over which consensus will be conducted is specified as the constant `VALUE_SET`, and the default value is specified as the constant `VALUE_0`. The initial value of the leader is set as `VALUE_1`. The `SynchronousExecutor` class runs the network for a series of five rounds. Each round gives each node the opportunity to send messages to its adjacent nodes and then to receive messages sent to them for processing. Then the test case checks whether agreement and validity are satisfied.

The project code provides a full implementation of the FollowLeader consensus protocol to help you understand how the testing framework works. The FollowLeader consensus protocol is in the `hw4.consensus.follow` package and consists of three files: `FollowLeaderNode`, `FollowLeaderPayload`, and `FollowLeaderMaliciousNode`. The `FollowLeaderNode` class is a compliant implementation of the FollowLeader protocol. The `FollowLeaderPayload` class represent the structure of the messages that are passed between nodes in the protocol. The `FollowLeaderMaliciousNode` class simulates an attacker that attempts to subvert the FollowLeader protocol.

The project code also gives incomplete implementations of the MajorityVoting and UnauthBFT consensus protocols. Your task is to complete them by making all the tests in hw4.BasicTests pass. You will very likely need to write your own tests to validate your work. The tests provided in the project are only given to you to clarify how the code should be written.

You may only modify the `UnauthBFTNode.java` and the `MajorityVotingMaliciousNode.java` classes. You may also submit your own helper classes that the grader will place under the `hw4.consensus.bft` package. No changes to any other project classes will be accepted as part of your submission.

Part 1: Attacking the Majority Voting Protocol

Give an implementation of a malicious node in the `MajorityVotingMaliciousNode.java` file that subverts the consensus of the MajorityVoting protocol that is implemented in the `MajorityVotingNode.java`. Note that you may need to leverage the ability of a malicious node to conspire with a few other malicious nodes. See `test3A` in `hw4.BasicTests`.

Part 2: Unauthenticated Byzantine Fault Tolerance

Give an implementation of the unauthenticated BFT consensus protocol [1] in the `UnauthBFTNode.java` file. You will likely need to create multiple helper Java classes. For example, you may want to create a helper classes that implements exponential information gathering tree data structure.

Frequently Asked Questions

1: For the malicious nodes in majority voting, is there an expected strategy we should use?

A strategy was briefly outlined during class, however you are free to devise your own strategy, as long as it breaks consensus consistently.

2. Can we use randomness, or do we need to think of a strategy for the malicious nodes so that they can try their best to break the consensus.

Your implementation may use randomness, but it should be deterministic in whether it actually breaks consensus. In other words, if it breaks during one run, it will break it on every subsequent run. Otherwise, it will be too difficult to grade.

3. My understanding is that the malicious node/leader can send anything regardless of what's actually initialized/sent to it. Is this correct?

Correct.

4. Is it guaranteed that the leader is malicious? Because if the leader is not malicious, there is no guarantee that the consensus can be break based on the number of malicious nodes.

You may assume that the test cases for grading will always make the leader malicious with at least 2 Sybil nodes. As the beginning of Lamport's paper explained v, the leader designation is only used in the subproblem to the actual problem of consensus in which there is no leader. In other words, if the malicious node breaks consensus when the leader is malicious, then it will also break it in the general case of consensus where there is no leader.

5. What will be the grading criteria for HW4? Except the 3 sample tests for regular majority voting, I see only 4 tests there and they are only checking two conditions (agreement and validity).

Part 1 will be graded based on whether the malicious node is able to break consensus when the leader is malicious and has access to at least two Sybil nodes. Consensus is broken when either agreement or validity is not reached.

Part 2 will be graded based on whether the compliant node is able to reach consensus consistently when at most a third of the nodes are malicious. Reaching consensus means that both agreement and validity are satisfied.

6. The project code has a lot of inline console outputs in the code. Will processing factor into grading?

Processing won't factor into the grading. Grading will be based on solely on whether the end result is breaking (Part 1) or reaching (Part 2) consensus properly.

References

1. Lamport, Shostak, Pease. *The Byzantine General's Problem*. ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, Pages 382-401.