Bitcoin and Cryptocurrency Technologies

Assignment 2: Consensus from trust

For this assignment, you will design and implement a distributed consensus algorithm in a graph framework that's somewhat similar to the Ripple network. A trust network is an alternative method of resisting sybil attacks and achieving consensus; it has the benefit of not wasting electricity like proof-of-work does.

You will write a CompliantNode class (which implements a provided Node class) that defines the behavior of a single node in a graph of around 100 nodes. The network of nodes is a directed random graph, where each edge represents a trust relationship. For example, if there is an A --> B edge, it means that Node B listens to transactions broadcast by node A. We say that B is A's follower and A is B's followee.

The provided Node class has the following API:

```
public interface Node {

	// boolean array of followees
	void setFollowees(boolean[] followees);

	// initialize proposal list of transactions
	void setPendingTransaction(ArrayList<Transaction> pendingTransactions);

	// return proposals to send to my followers. After final round, should return
	// the transactions upon which consensus has been reached.
	Set<Transaction> getProposals();

	// receive candidates from other nodes.
	void receiveCandidates(ArrayList<Integer[]> candidates);
}
```

Each node will know its followees via a boolean array whose indices correspond to nodes in the graph. A 'true' in index i indicates that node i is a followee, 'false' otherwise. That node will also have a list of transactions (its proposal list) that it can broadcast to its followers. Generating the initial transactions/proposal list will not be your responsibility. Assume that all transactions are valid and that invalid transactions cannot be created.

Each node should succeed in achieving consensus with a network in which its peers are other nodes running the same code. Your algorithm should be designed such that a network of nodes receiving different sets of transactions can agree on a set to be accepted. We will be providing a Simulation.java file that generates a random trust graph. There will be a set number of rounds where during each

round, your nodes will broadcast their proposal to their followers and at the end of the round, should have reached a consensus on what transactions should be agreed upon.

In testing, the nodes running your code may encounter a number (up to 45%) of malicious nodes that do not cooperate with your consensus algorithm. Nodes of your design should be able to withstand as many malicious nodes as possible and still achieve consensus. Malicious nodes may have arbitrary behavior. For instance, among other things, a malicious node might:

- be functionally dead and never actually broadcast any transactions.
- constantly broadcasts its own set of transactions and never accept transactions given to it.
- change behavior between rounds to avoid detection.

You will be provided the following files:

Node.java	a basic interface for your CompliantNode class
MaliciousNode.java	a very simple example of a malicious node
Simulation.java	a basic graph generator that you may use to run your own simulations with varying graph parameters (described below) and test your CompliantNode class
Transaction.java	the transaction class, so you know we are not doing anything shady

The graph of nodes will tentatively have the following parameters:

- the pairwise connectivity probability of the random graph: e.g. {.1, .2, .3}
- the probability that a node will be set to be malicious: e.g {.15, .30, .45}
- the probability that each of the initial valid transactions will be communicated: e.g. {.01, .05, .10}
- the number of rounds in the simulation e.g. {10, 20}

Your focus will be on developing a robust CompliantNode class that will work in all 3x3x3x2 = 54 combinations of the graph parameters. At the end of each round, your node will see the list of transactions that were broadcast to it.

A test is considered successful if

- it achieves 100% consensus. This means that **all compliant nodes must output the same lists**. There is no such thing as a 85% success rate in this case: consensus will be considered binary. All compliant must return the same lists for the test to succeed.
- Execution time is within reason.

Note that you can make a test succeed by simply returning an empty set, but this is not a good consensus algorithm. You should strive to make the consensus set of transactions as large as possible.

Some Hints:

- Your node will not know the network topology and should do its best to work in the general case. That said, be aware of how different topology might impact how you want to include transactions in your picture of consensus.
- All nodes running your consensus algorithm should ultimately be able to produce the same list of transactions
- Your CompliantNode code can assume that all Transactions it sees are valid -- the simulation code will only send you valid transactions (both initially and between rounds) and only the simulation code has the ability to create valid transactions.
- Ignore pathological cases that occur with extremely low probability, for example where a compliant node happens to pair with only malicious nodes. We will make sure that the actual tests cases do not have such scenarios.

Extra credit:

- Perform an analysis of the maximum percentage of malicious nodes that can be tolerated as a function of the parameters of the random graph, the number of rounds allowed, and any other parameters that you think are relevant.